

CS 50 Web Design

APCSP Module 2: Internet



Unit 3: JavaScript

LECTURE 8: PROGRAM STRUCTURES

DR. ERIC CHOU

IEEE SENIOR MEMBER



Objectives

- Program structures in JavaScript include the following: conditional statements, switch, for-loop, while-loop, do-while loop, and functions.
- Break levels for loops and functions.
- Random number generation
- Brief introduction for functional programming



Objectives

- By the end of the lesson, you will be familiar and know how the website works using JavaScripts.
 - Discuss the introduction to JavaScript and using conditional statements.
 - Understand the coding syntax using the break keywords.
 - Explain thoroughly the coding styles of different kinds of loops.

Conditionals and Logic

SECTION 1

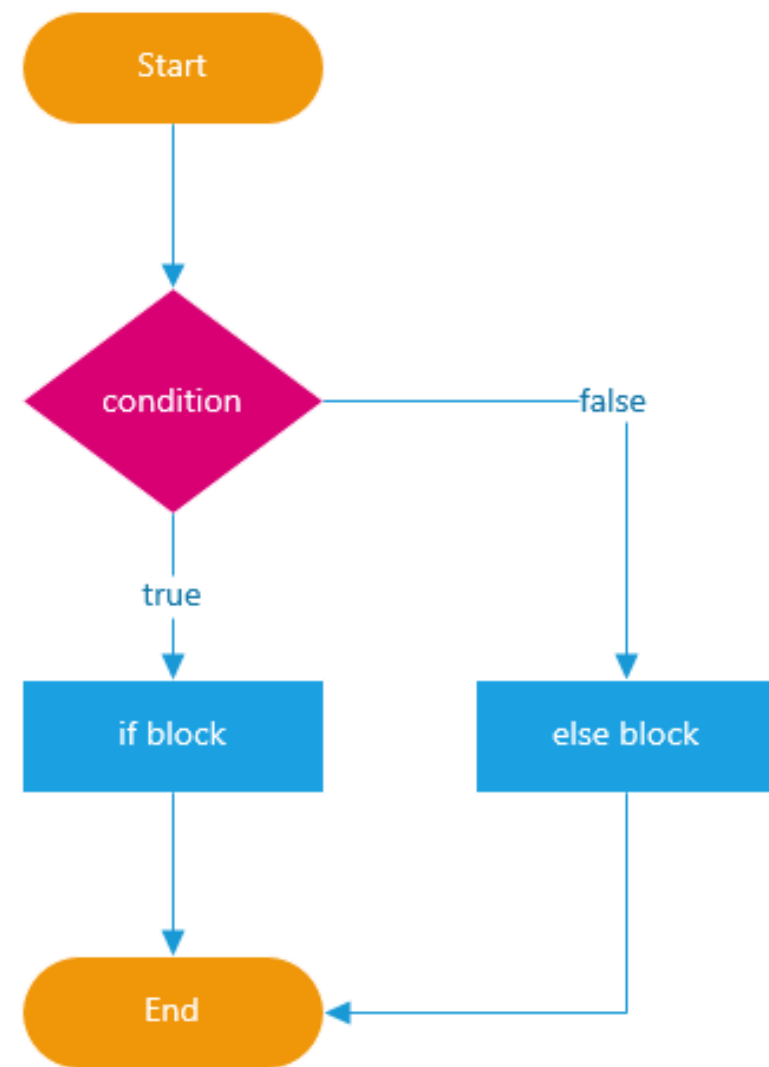
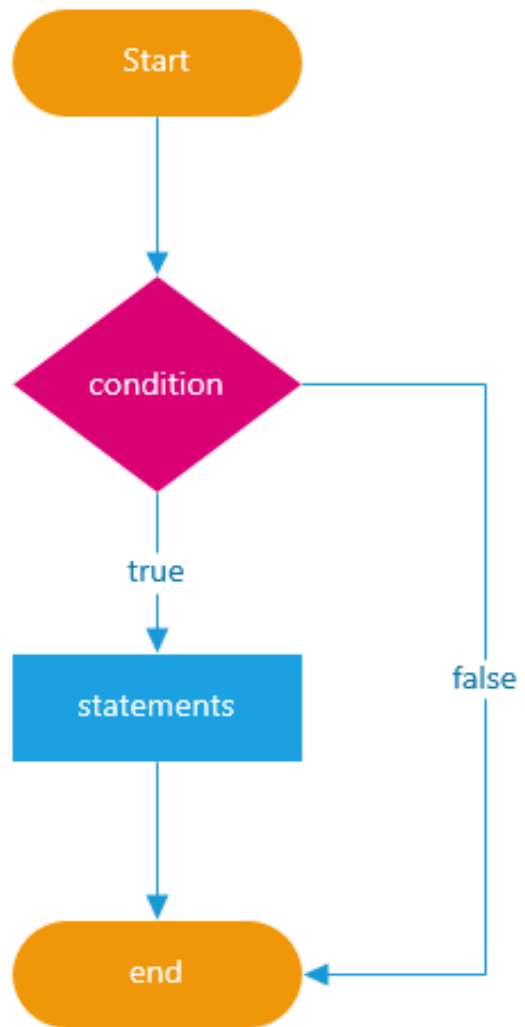
Conditionals and Logic

Conditionals and Logic are used to perform different actions based on different conditions.

- Very often when we write code, we wanted to perform different actions for different decisions.
- We can use Conditionals and Logic in our code to do this.

Conditionals and Logic

- In JavaScript, we have the following conditional statements:
 - Use **if** to specify a block of code to be executed, if a specified condition is true
 - Use **else** to specify a block of code to be executed, if the same condition is false.
 - Use **else if** to specify a new condition to test, if the first condition is false.
 - Use **switch** to specify many alternative blocks of code to be executed.



The if Statement

- Use **if** to specify a block of code to be executed, if a specified condition is true.

- **Syntax:**

```
if (condition) {  
    block of code to be executed  
if  
    the condition is true  
}
```

- Note that **if** is in lower case. Uppercase letters will generate a JavaScript error.



The if Statement

- Example:

Make a “Good day” greeting if the hour is less than 18:00:

```
if (hour < 18) {  
    greeting = "Good day";  
}
```

- The result of greeting will be:
Good day

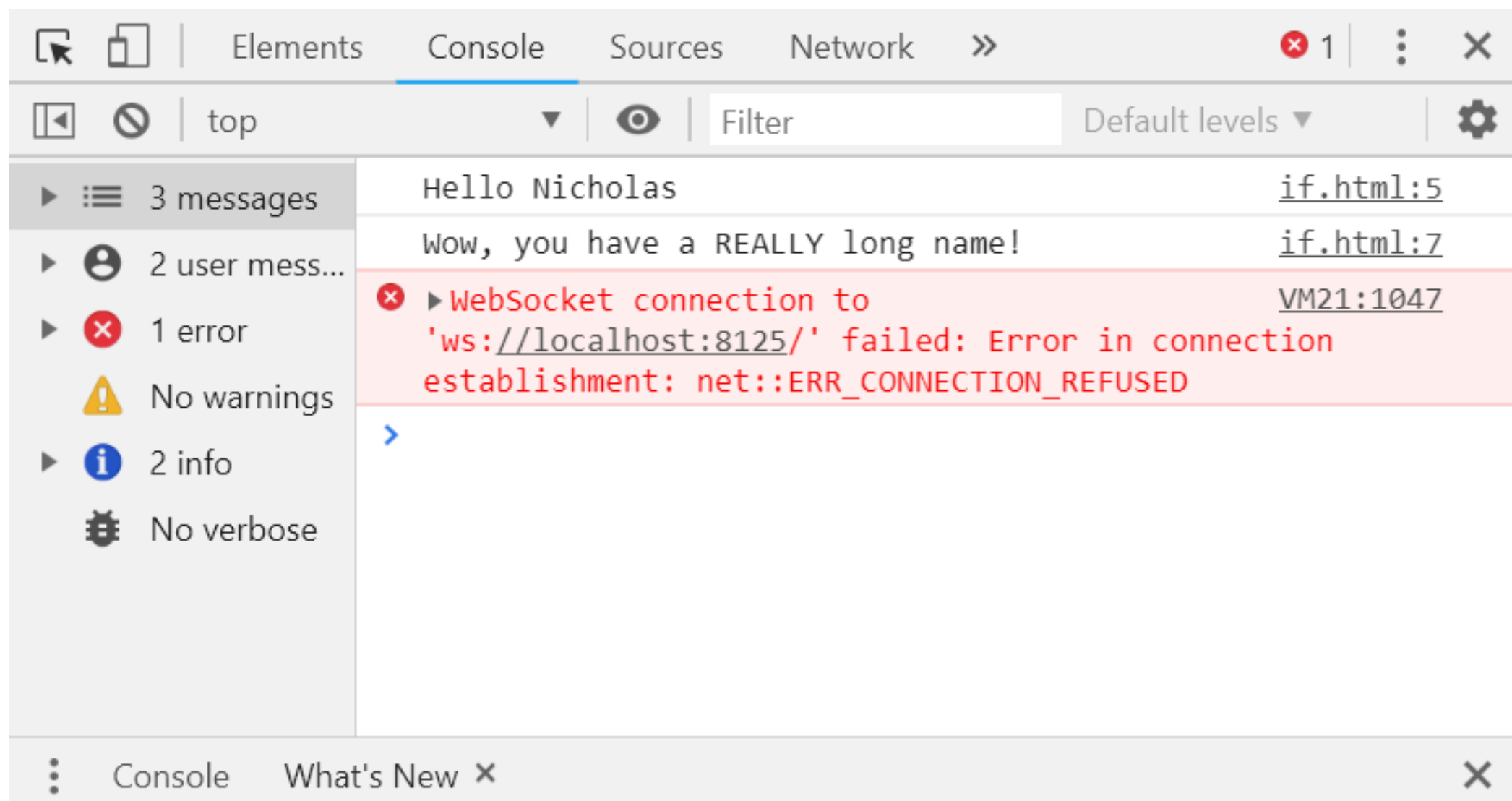
The else Statement

- Use **else** to specify a block of code to be executed, if the same condition is false.

```
if (condition) {  
    block of code to be executed  
    if the condition is true  
}  
else {  
    block of code to be executed  
    if  
    the condition is false  
}
```

```
1 ▼ <html>
2 ▼   <body>
3 ▼   <script>
4       var name = "Nicholas";
5       console.log("Hello " + name);
6 ▼       if (name.length > 7) {
7           console.log("Wow, you have a REALLY long name!");
8       }
9   </script>
10  </body>
11 </html>
```

Demo Program: if.html



The else Statement

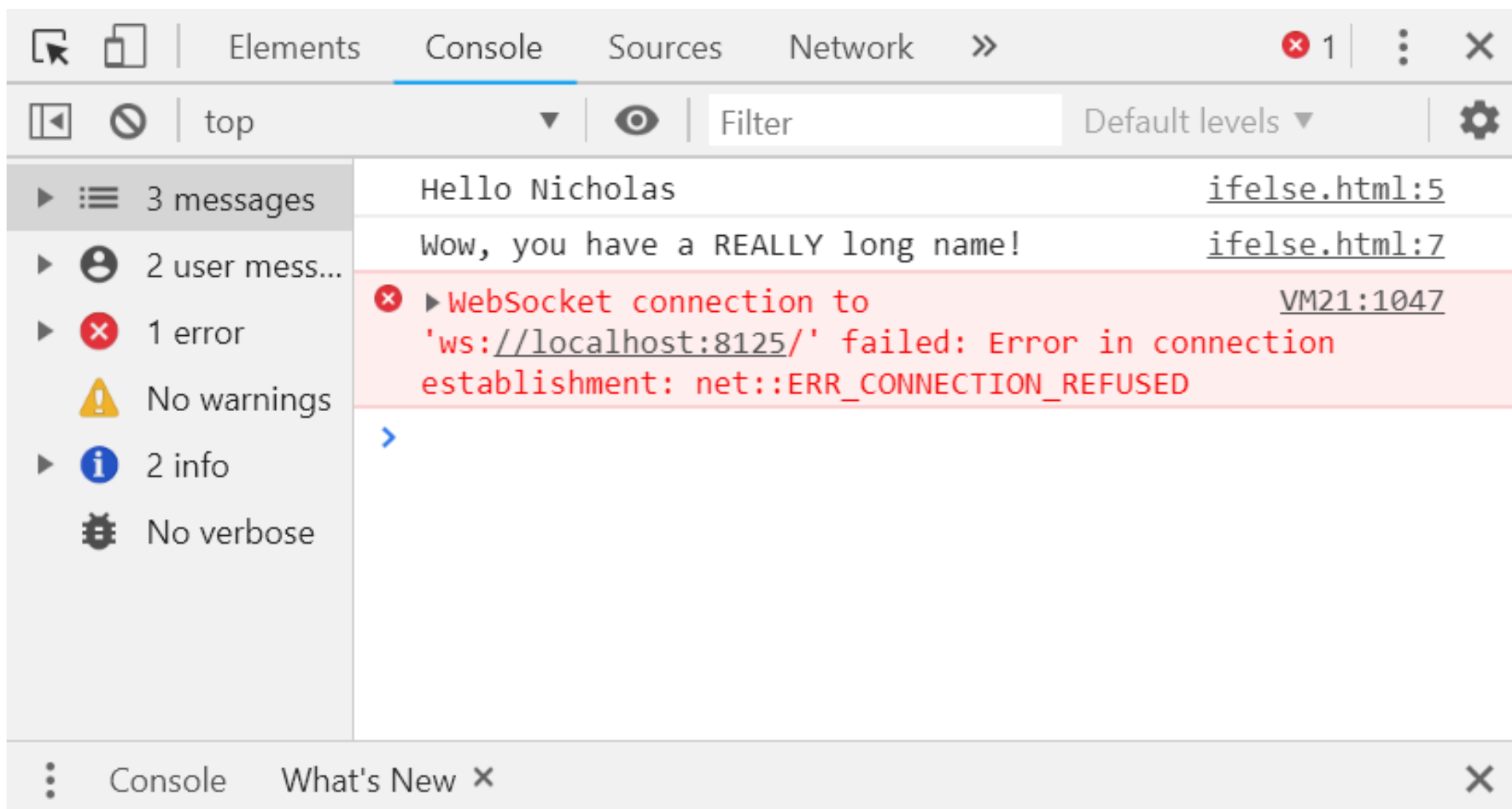
- **Example:**
- If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```
- The result of greeting will be:
Good day



Demo Program: ifelse.html

```
1 ▼ <html>
2 ▼   <body>
3 ▼   <script>
4       var name = "Nicholas";
5       console.log("Hello " + name);
6 ▼     if (name.length > 7) {
7         console.log("Wow, you have a REALLY long name!");
8     }
9 ▼     else {
10        console.log("Your name isn't very long.");
11    }
12    </script>
13    </body>
14 </html>
```



The else if Statement

- Use **else if** to specify a new condition to test, if the first condition is false.

```
if (condition1) {  
    block of code to be executed if  
    condition1 is true  
} else if (condition2) {  
    block of code to be executed if  
    the condition1 is false and  
    condition2 is true  
} else {  
    block of code to be executed if  
    the condition1 is false and  
    condition2 is false  
}
```




The else if Statement

- **Example:**

- If time is less than 10:00, create a “Good morning” greeting, if not, but time is less than 20:00, create a “Good day” greeting, otherwise a “Good evening”:

```
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

Demo Program:
ifelseChained.html

- Nested if-else-if-else statements.
- Maybe replaced by switch-statement in some situations.

```
1 ▼ <html>
2 ▼   <body>
3 ▼   <script>
4       var lemonChicken = false;
5       var beefWithBlackBean = true;
6       var sweetAndSourPork = true;
7 ▼     if (lemonChicken) {
8         console.log("Great! I'm having lemon chicken!");
9 ▼     } else if (beefWithBlackBean) {
10         console.log("I'm having the beef.");
11 ▼     } else if (sweetAndSourPork) {
12         console.log("OK, I'll have the pork.");
13 ▼     } else {
14         console.log("Well, I guess I'll have rice then.");
15     }
16   </script>
17 </body>
18 </html>
```

127.0.0.1:57436/pass/iffelseChained x +

127.0.0.1:57436/pass/iffelseChained.html

Apps

Elements Console Sources Network Performance >> 1

top Filter Default levels

2 messages

1 user mess...

1 error

No warnings

1 info

No verbose

I'm having the beef. iffelseChained.html:10

Failed to load resource: the server responded with a status of 404 (Not Found) :57436/favicon.ico:1

Console What's New

Highlights from the Chrome 74 update

Switch

SECTION 2

The switch Statement

- Use **switch** to specify many alternative blocks of code to be executed.

- Syntax:

```
switch(expression) {  
    case n:    /* code block */  
        break;  
    case n:    /* code block */  
        break;  
    default: /* default code  
            block */  
}
```

The switch Statement

- How switch works?
- The switch expression is evaluated once.
- The value of the expression is compared with the value of each case.
- If there is a match, the associated block of code is executed.

The `switch` Statement

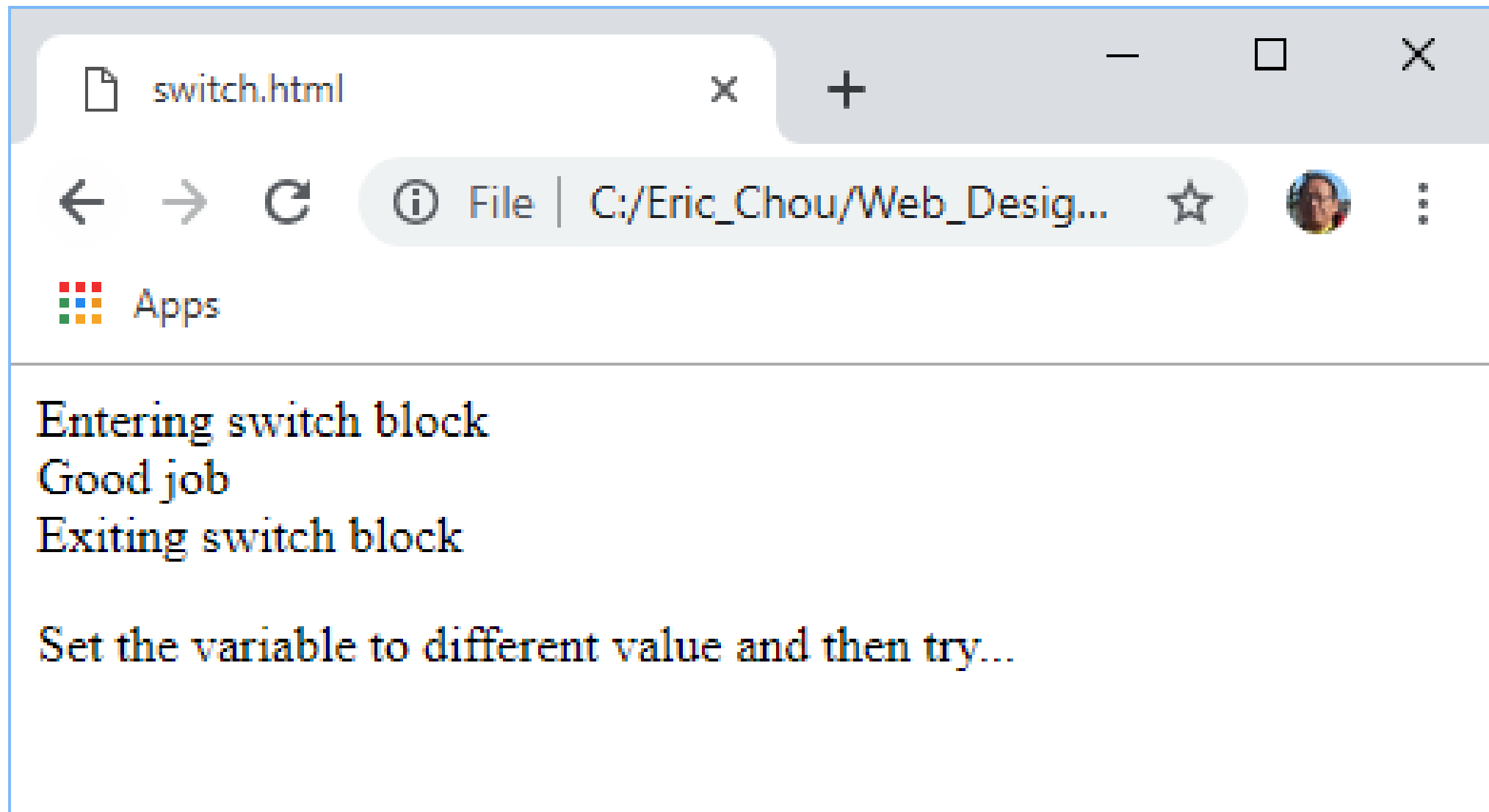
- Example:
 - The `getDay()` method returns the weekday as a number between 0 and 6. (Sunday=0, Monday=1, Tuesday=2 ..)
 - Use the weekday number to calculate weekday name.

Switch

Demo Program: switch.html

- 6-way switch.
- String as switch variable
- Output to document.

```
1 ▼ <html>
2 ▼   <body>
3 ▼     <script type = "text/javascript">
4       var grade = 'A';
5       document.write("Entering switch block<br />");
6 ▼     switch (grade) {
7       case 'A': document.write("Good job<br />");      break;
8       case 'B': document.write("Pretty good<br />");  break;
9       case 'C': document.write("Passed<br />");        break;
10      case 'D': document.write("Not so good<br />");   break;
11      case 'F': document.write("Failed<br />");        break;
12      default:  document.write("Unknown grade<br />")
13    }
14    document.write("Exiting switch block");
15  </script>
16  <p>Set the variable to different value and then try...</p>
17 </body>
18 </html>
```



Conditional Operators

SECTION 3



JavaScript if else shortcut: conditional operator

- JavaScript provides a conditional operator that can be used as a shortcut of the if statement. The following illustrates the syntax of the conditional operator.

condition ? expression_1 : expression_2

- Like the if statement, the condition is an expression that evaluates to true or false. If the condition evaluates to true, the operator returns the value of the expression_1; otherwise, it returns the value of the expression_2.



JavaScript if else shortcut: conditional operator

- For example, to display a different label for the login button based on the value of the `isLoggedIn` variable, you could use the conditional operator as follows:

```
isLoggedIn ? "Logout" : "Login";
```

- You can also assign a variable depending on the result of the ternary operator.

```
// only register if the age is greater than 18  
var allowRegister = age > 18 ? true : false;
```



JavaScript if else shortcut: conditional operator

- If you want to do more than a single operation per case, you need to separate operation using a comma (,) as the following example:

```
age > 18 ? (  
    alert("OK, you can register."),  
    redirectTo("register.html");  
) : (  
    stop = true,  
    alert("Sorry, you are too young!")  
);
```

- In this tutorial, you have learned how to use the JavaScript if else statement to execute a statement when a condition evaluates to true and executes another statement when the condition evaluates to false.

Loops

SECTION 4

Different Kinds of Loops

- JavaScript supports different kinds of loops:
 - **for** – loops through a block of code a number of times.
 - **for/in** – loops through the properties of an object.
 - **while** – loops through a block of code while a specified condition is true.
 - **do/while** - also loops through a block of code while specified condition is true.

JavaScript **for** loop

- Loops can execute a block of code a number of times.
- Loops are handy, if you want to run the same code over and over again, each time with a different value. Often this is the case when working with arrays:

```
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";
```

JavaScript for loop

- The for loop is often the tool you will use when you want to create a loop.
- The for loop has the following syntax:

```
for(statement1; statement2; statement3) {  
    code block to be executed  
}
```

JavaScript for loop

- Often this is the case when working with arrays:
- We can write:

```
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";
```

```
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}
```

JavaScript for loop

- Statement1 is executed before the loop (the code block) starts.
- Statement2 defines the condition running the loop (the code block)
- Statement3 is executed each time after the loop (the code block) has been executed.
- **Example:**

```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

JavaScript **for/in** loop (for-each- attribute)

- The JavaScript **for/in** statement loops through the properties of an object:

```
var person = {fname:"John",  
              lname:"Doe", age:25};  
  
var text = "";  
var x;  
  
for (x in person) {  
    text += person[x];  
}
```

JavaScript **while** loop

- The while loop loops through a block of code as long as a specified condition is true.
- **Syntax:**

```
while (condition) {  
  code block to be executed  
}
```

JavaScript **while** loop

- **Example:**
- The code in the loop will run over and over again, as long as variable (i) is less than 10.

```
while (i < 10) {  
    text += "The number is " +  
    i;  
    i++;  
}
```


JavaScript do/while loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

- **Syntax:**

```
do {  
    code block to be executed  
}  
  
while (condition);
```

JavaScript do/while loop

- **Example:**
- The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

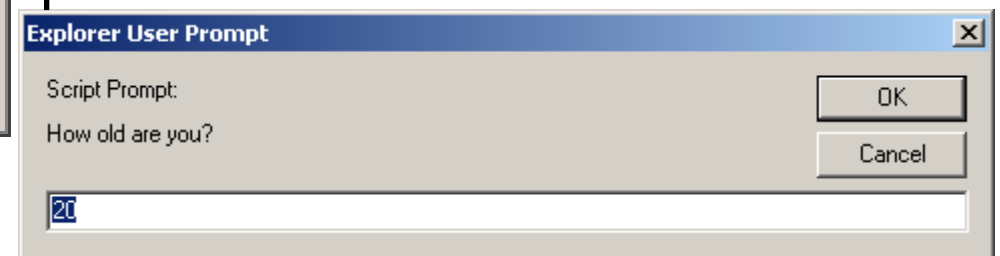
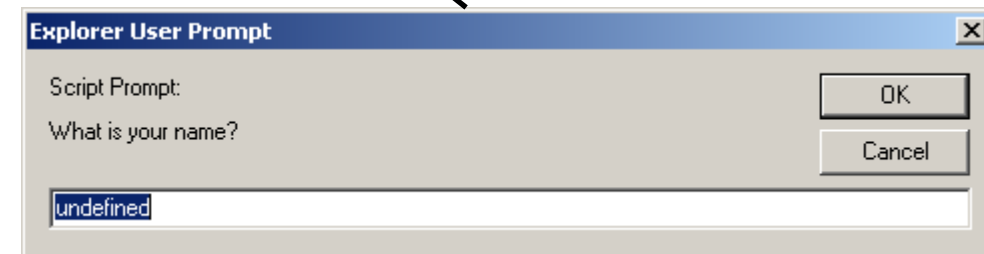
```
do {  
    text += "The number is "  
           + i;  
    i++;  
}  
while (i < 10);
```

Input/Output with Browsers

SECTION 5

alert(),
confirm(),
and **prompt()**

```
<script type="text/javascript">  
alert("This is an Alert method");  
confirm("Are you OK?");  
prompt("What is your name?");  
prompt("How old are you?", "20");  
</script>
```





alert() and confirm()

```
alert("Text to be displayed");
```

- Display a message in a dialog box.
- The dialog box will block the browser.

```
var answer = confirm("Are you sure?");
```

- Display a message in a dialog box with two buttons: "OK" or "Cancel".
- `confirm()` returns **true** if the user click "OK". Otherwise it returns **false**.



prompt ()

```
prompt("What is your student id number?");  
prompt("What is your name?", "No name");
```

- Display a message and allow the user to enter a value
- The second argument is the "default value" to be displayed in the input textfield.
- Without the default value, "undefined" is shown in the input textfield.
- If the user click the "OK" button, **prompt ()** returns the value in the input textfield as a string.
- If the user click the "Cancel" button, **prompt ()** returns null.

Break Levels

SECTION 6



break

The **break** Keyword

- When the JavaScript code interpreter reaches a **break** keyword, it breaks out of the switch block.
 - This will stop the execution of more code and case testing inside the block.
- Note: When a match is found, and the job is done, it is time for a break. There is no need for more testing.



Label statement

- Before discussing the break statement, let's talk about the label statement first.
- In JavaScript, you can label a statement for later use. The following illustrates the syntax of the label statement.

```
1 label: statement;
```

- The label can be any valid identifier.
- The following example labels the loop using the outer label.

```
1 outer: for (var i = 0; i < 5; i++) {  
2     console.log(i);  
3 }
```

- You can reference to the label by using the break or continue statement. Typically, you use the label with nested loop such as for, do-while, and while loop.



JavaScript break statement

Demo Program: [break.html](#)

- The break statement gives you a fine-grained control over the execution of the code in a loop. The break statement terminates the loop immediately and passes control over the next statement after the loop.
- Here's an example.

```
1 for (var i = 1; i < 10; i++) {  
2     if (i % 3 == 0) {  
3         break;  
4     }  
5 }  
6 console.log(i); // 3
```

JavaScript break statement

- In this example, the `for` loop increments the variable `i` from 1 to 10. In the body of the loop, the `if` statement checks if `i` is evenly divisible by 3. If so, the `break` statement is executed and the loop is terminated.
- The control is passed to the next statement outside the loop that outputs the variable `i` to the web console.
- Besides controlling the loop, you also use the `break` statement to terminate a case branch in the `switch` block.

Elements Console Sources Network >> 1

top Filter Default levels

Hide console sidebar

- ▶ 1 user mess...
- ▶ 1 error
- ! No warnings
- ▶ 1 info
- ⚙ No verbose

3 break.html:9

✖ ▶ WebSocket connection to 'ws://localhost:8125/' failed: Error in connection establishment: net::ERR_CONNECTION_REFUSED VM21:1047

>

Using **break** statement to exit nested loop

- As mentioned earlier, you use the break statement to terminate a label statement and transfer control to the next statement following the terminated statement. The syntax is as follows:

```
break label;
```



Using **break** statement to exit nested loop

Demo Program: [break2.html](#)

- The `break` statement is typically used to exit the nested loop. See the following example.

```
1 var iterations = 0;
2 top: for (var i = 0; i < 5; i++) {
3     for (var j = 0; j < 5; j++) {
4         iterations++;
5         if (i === 2 && j === 2) {
6             break top;
7         }
8     }
9 }
10 console.log(iterations); // 13
```

Elements Console Sources Network >> 1

top Filter Default levels

2 messages

1 user mess...

1 error

No warnings

1 info

No verbose

13 break2.html:13

✖ ▶ WebSocket connection to VM21:1047
'ws://localhost:8125/' failed: Error in connection
establishment: net::ERR_CONNECTION_REFUSED

> |



continue

JavaScript **continue**: Skips the Current Iteration of a Loop

- **Summary:** in this tutorial, you will learn how to use the JavaScript `continue` statement to skip the current iteration of a loop.
- The **`continue`** statement skips the current iteration of a loop and goes to the next one. Because of this, the `continue` statement must appear in the body of a loop or you will get an error.
- Similar to the **`break`** statement, the `continue` statement has two forms: labeled and unlabeled. For more information on the label statement, see the `break` statement tutorial.



Using unlabeled JavaScript **continue** statement

- The unlabeled continue statement skips the current iteration of a **for**, **do-while**, or **while loop**. The continue statement skips the rest of the code to the end of the innermost body of a loop and evaluates the expression that controls the loop.



Using unlabeled JavaScript **continue** statement

- In a for loop, the continue skips all the statements underneath it and pass the execution of the code to the update expression, in this case, it is **i++**;

```
1  for (var i = 0; i < count; i++) {  
2      if (condition)  
3          continue; // Jumps to expression: i++  
4      // more statement here  
5  }
```



Using unlabeled JavaScript `continue` statement

- In a while or do-while loop, it jumps back to the expression that controls the loop.

```
1 while (expression) // continue jumps here
2 {
3     if (condition) {
4         continue; // Jumps to expression
5     }
6     // more statements here
7     // ...
8 }
```

```
1 do{
2     if (condition) {
3         continue; // Jumps to expression
4     }
5     // more statements here
6     // ...
7 }while(expression); // continue jumps here
```



Using unlabeled JavaScript **continue** statement

Demo Program: continue.html (3)

```
1 var s = 'This is a JavaScript continue statement demo.';
2 var counter = 0;
3 for (var i = 0; i < s.length; i++) {
4     if (s.charAt(i) != 's') {
5         continue;
6     }
7     //
8     counter++;
9 }
10 console.log('The number of s found in the string is ' + counter);
```



Using JavaScript `continue` with a label

The `continue` statement can include an optional label as follows:

```
1 continue label;
```

The `label` can be any valid identifier.

See the following example.

```
1 // continue with a label
2 outer: for (var i = 1; i <= 3; i++) {
3     for (var j = 1; j <= 3; j++) {
4         if ((i == 2) && (j == 2)) {
5             console.log('continue to outer');
6             continue outer;
7         }
8         console.log("[i:" + i + ",j:" + j + "]");
9     }
10 }
```



pass in
JavaScript

```
1 ▼ <html>
2 ▼   <body>
3 ▼   <script>
4     var i=0;
5       document.write("<h1>Demo of Pass: </h1>");
6       // single ; can be used for pass
7 ▼     if (i==0) {
8         ;
9     }
10    document.write("Passed...");
11  </script>
12  </body>
13 </html>|
```

pass in JavaScript

[Demo Program: pass.html](#)

USE SINGLE ; SYMBOL AS
PASS STATEMENT;



return

Functions

SECTION 12



The Basic Anatomy of a Function

- Figure below shows how a function is built. The code between the curly brackets is called the *function body*, just as the code between the curly brackets in a loop is called the *loop body*.

```
function () {  
    console.log("Do something");  
}
```

↖
The function body
goes between curly brackets.

Creating a Simple Function

- Let's create a simple function that prints Hello world!. Enter the following code in the browser
- console. Use SHIFT-ENTER to start each new line without executing the code.

```
var ourFirstFunction =  
function () {  
  console.log("Hello world!");  
};
```
- This code creates a new function and saves it in the variable **ourFirstFunction**.

Calling a Function

- To run the code inside a function (the function body), we need to call the function. To call a function, you enter its name followed by a pair of opening and closing parentheses, as shown here.

```
> ourFirstFunction();  
Hello world!
```

- Calling **ourFirstFunction** executes the body of the function, which is **console.log("Hello world!");**, and the text we asked to be printed is displayed on the next line: **Hello world!** .
- But if you call this function in your browser, you'll notice that there's a third line, with a little leftfacing arrow, as shown in Figure 8-2. This is the return value of the function.



Functional Call

```
> ourFirstFunction();  
Hello, world!  
← undefined
```

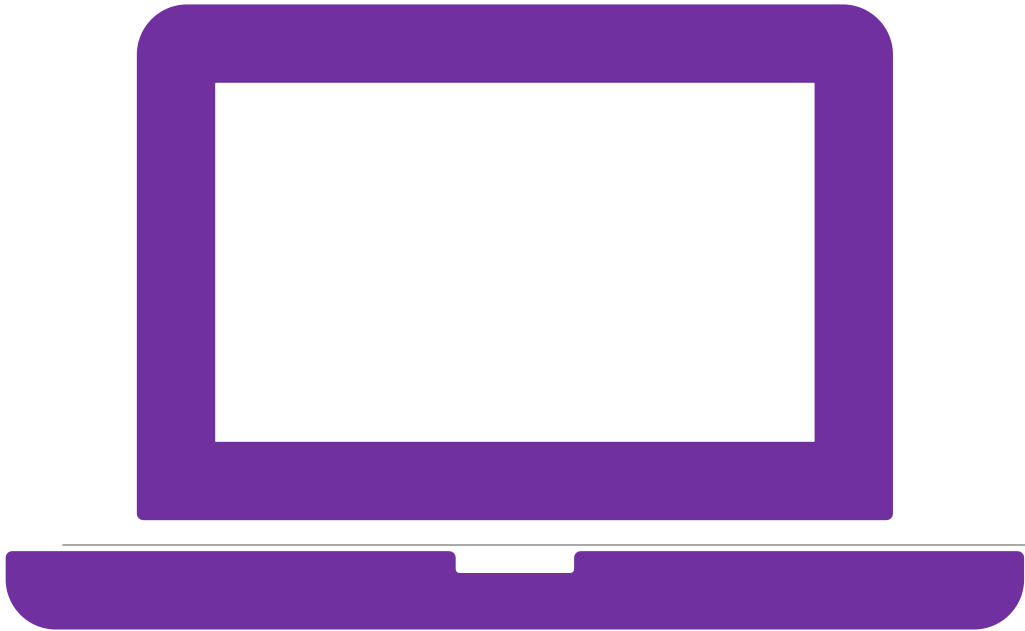
Figure 8-2. Calling a function with an undefined return value

Functional Call with undefined Return Value

- A **return** value is the value that a function outputs, which can then be used elsewhere in your code.
- In this case, the return value is undefined because we didn't tell the function to return any particular value in the body of the function.

Functional Call with undefined Return Value

- All we did was ask it to print a message to the console, which is not the same as returning a value.
- A function always returns **undefined** unless there is something in the function body that tells it to return a different value. (We'll look at how to specify a return value in Returning Values from Functions.)



In-Class Demonstration Program

- Function returning undefined

FUNCTION2.HTML

Parameters

SECTION 13



Passing Arguments into Functions

- **ourFirstFunction** just prints the same line of text every time you call it, but you'll probably want your functions to be more flexible than that.
- Function *arguments* allow us to pass values into a function in order to change the function's behavior when it's called.
- Arguments always go between the function parentheses, both when you create the function and when you call it.

Passing Arguments into Functions

- The following **sayHelloTo** function uses an argument (name) to say hello to someone you specify.

```
var sayHelloTo =  
function (name) {  
    console.log("Hello " +  
name + " !");  
};
```

Passing Arguments into Functions

- We create the function in the first line and assign it to the variable **sayHelloTo**. When the function is called, it logs the string “Hello ” + name + “!”, replacing name with whatever value you pass to the function as an argument.
- Figure 8-3 shows the syntax for a function with one argument.

An argument name



```
function ( argument ) {  
    console.log("My argument was: " + argument);  
}
```



This function body can
use the argument.

Figure 8-3. The syntax for creating a function with one argument



Passing Arguments into Functions

- To call a function that takes an argument, place the value you'd like to use for the argument between the parentheses following the function name. For example, to say hello to Nick, you would write:

```
sayHelloTo("Nick") ;
```

```
Hello Nick!
```

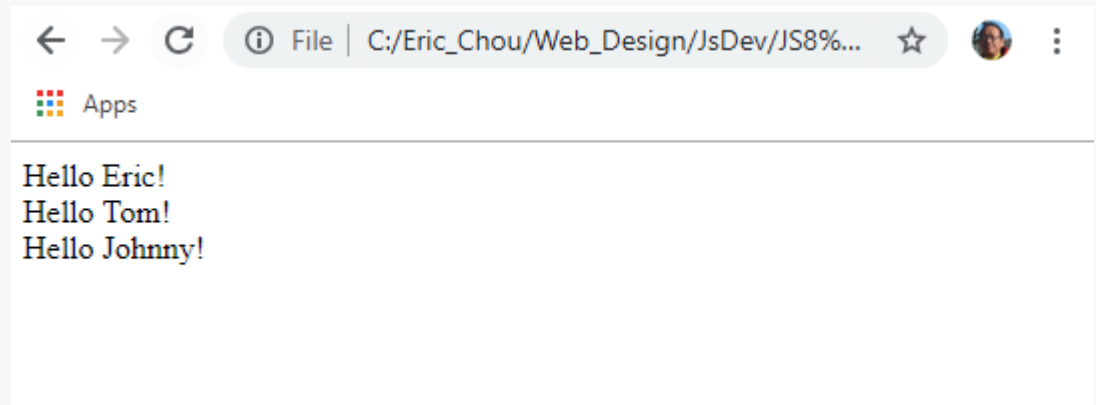

Passing Arguments into Functions

- Or, to say hello to Lyra, write:
`sayHelloTo("Lyra");`
`Hello Lyra!`
- Each time we call the function, the argument we pass in for name is included in the string printed by the function. So when we pass in “Nick”, the console prints “Hello Nick!”, and when we pass in “Lyra”, it prints "Hello Lyra!".



Demo Program: sayHello.html

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼   var sayHelloTo = function (name) {
5     document.write("Hello " + name + "<br>");
6   };
7   sayHelloTo("Eric");
8   sayHelloTo("Tom");
9   sayHelloTo("Johnny");
10  </script>
11  </body>
12  </html>
```



Printing Cat Faces!

```
var drawCats = function (howManyTimes) {  
  for (var i = 0; i < howManyTimes; i++) {  
    console.log(i + " = ^.^=");  
  }  
};
```



Printing Cat Faces!

Demo Program: `printCats.html`

`drawCats (5) ;`

0 `=^ . ^=`

1 `=^ . ^=`

2 `=^ . ^=`

3 `=^ . ^=`

4 `=^ . ^=`

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼ var drawCats = function (howManyTimes){
5 ▼ for (var i = 0; i < howManyTimes; i++){
6     document.write(i + " ^= ^=<br>");
7 }
8 };
9 drawCats(5);
10 </script>
11 </body>
12 </html>
```

Multiple Arguments

SECTION 14



Passing Multiple Arguments to a Function

Each argument name is
separated by a comma.



```
function (argument1, argument2) {  
  console.log("My first argument was: " + argument1);  
  console.log("My second argument was: " + argument2);  
}
```



The function body can
use both arguments.

Passing Multiple Arguments to a Function

- The following function, **printMultipleTimes**, is like `drawCats` except that it has a second argument called **whatToDraw**.

```
var printMultipleTimes = function
(howManyTimes, whatToDraw) {
  for (var i = 0; i < howManyTimes; i++) {
    console.log(i+" "+whatToDraw);
  }
};
```



Multiple Parameters

Demo Program: printManyTimes.html

```
> printMultipleTimes(5, "=^.^="); printMultipleTimes(4, "^_^");  
0 =^.^=  
1 =^.^=  
2 =^.^=  
3 =^.^=  
4 =^.^=  
0 ^ _ ^  
1 ^ _ ^  
2 ^ _ ^  
3 ^ _ ^
```



```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼ var printMultipleTimes = function (howManyTimes, whatToDraw){
5 ▼   for (var i = 0; i < howManyTimes; i++){
6       document.write(i+" "+whatToDraw+"<br>");
7   }
8 };
9   printMultipleTimes(5, "=^.^=");
10  printMultipleTimes(4, "^_^");
11 </script>
12 </body>
13 </html>
```

Return Value

SECTION 15

Returning Values from Functions

- The output of a function is called the return value. When you call a function that returns a value, you can use that value in the rest of your code (you could save a return value in a variable, pass it to another function, or simply combine it with other code).
- For example, the following line of code adds 5 to the return value of the call to `Math.floor(1.2345)`:

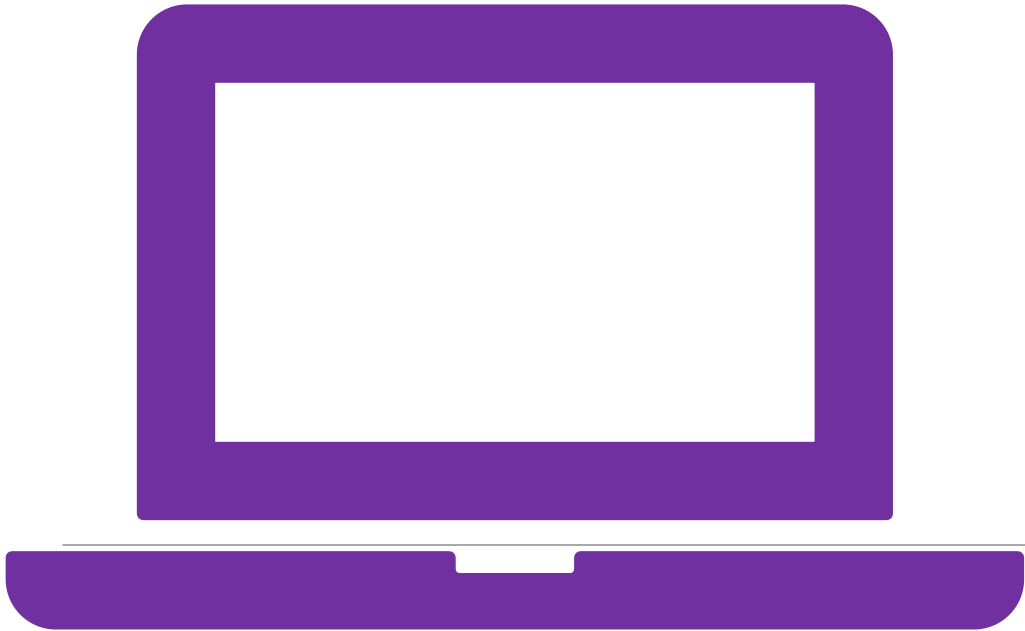

```
> 5 + Math.floor(1.2345) ;
```

6

Define a Function with Return Value

- Let's create a function that returns a value. The function `double` takes the argument `number` and returns the result of `number * 2`. In other words, the value returned by this function is twice the number supplied as its argument.

```
var double = function (number) {  
    return number * 2;  
};  
> double(3);  
6
```



In-Class Demonstration Program

- Dice (6 faces)
- Dice (multiple faces, base)

DICE.HTML

Function Call as value

SECTION 16

Using Function Calls as Values

- You can also pass a function call into another function as an argument, and the function call will be substituted with its return value.
- In this next example we call `double`, passing the result of calling `double` with 3 as an argument. We replace `double(3)` with 6 so that `double(double(3))` simplifies to `double(6)`, which then simplifies to 12.

```
> double(double(3)) ;
```

```
12
```

`double(double(3));`

① `double(3 * 2)`

② `double(6)`

③ `6 * 2`

④ `12`

Procedural Abstraction [Optional]

SECTION 17

Procedural Abstraction

- Procedural abstraction is the idea that each method should have a coherent conceptual description that separates its implementation from its users.
- You can encapsulate behavior in methods that are internal to an object or methods that are widely usable.

Using Functions to Simplify Code

In Chapter 3, we used the methods **`Math.random`** and **`Math.floor`** to pick random words from arrays and generate random insults. In this section, we'll re-create our insult generator and simplify it by creating functions.



A Function to Pick a Random Word

PART 1

A Function to Pick a
Random Word
Demo Program:
randomWords.html

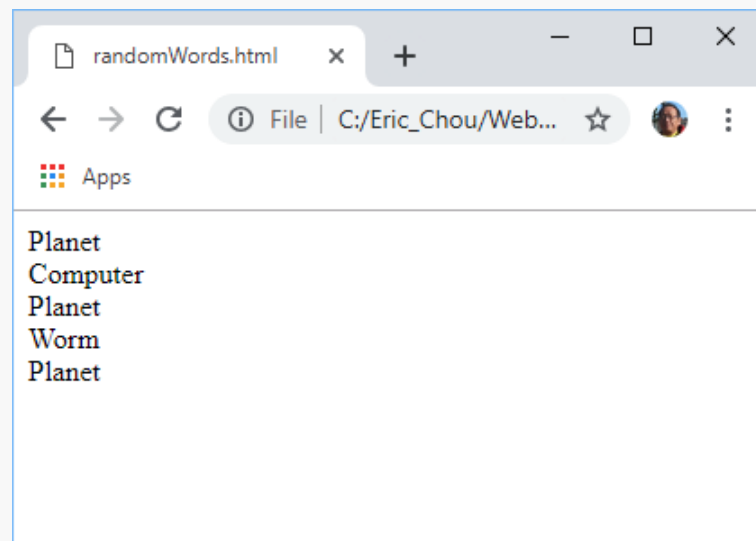
```
var randomWords = ["Planet", "Worm",  
                   "Flower", "Computer"];  
var pickRandomWord = function (words) {  
    return words[Math.floor(Math.random()  
        * words.length)];  
};  
  
> pickRandomWord(randomWords);  
"Flower"
```

A Function to Pick a Random Word

- We can use this same function on any array. For example, here's how we would pick a random name from an array of names:

```
> pickRandomWord(["Charlie",  
"Raj", "Nicole", "Kate",  
"Sandy"]);  
  
"Raj"
```

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4   var randomWords = ["Planet", "Worm", "Flower", "Computer"];
5 ▼ var pickRandomWord = function (words) {
6     return words[Math.floor(Math.random() * words.length)];
7 };
8 ▼ for (var i=0; i<5; i++){
9     var w = pickRandomWord(randomWords);
10    document.write(w+"<br>");
11 }
12 </script>
13 </body>
14 </html>
```





A Random Insult Generator

PART 2


```
var randomBodyParts = ["Face", "Nose", "Hair"];
var randomAdjectives = ["Smelly", "Boring", "Stupid"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];

// Pick a random body part from the randomBodyParts array:
var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
// Pick a random adjective from the randomAdjectives array:
var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
// Pick a random word from the randomWords array:
var randomWord = randomWords[Math.floor(Math.random() * 5)];
// Join all the random strings into a sentence:
var randomString = "Your " + randomBodyPart + " is like a " + 
randomAdjective + " " + randomWord + "!!!";
randomString;
"Your Nose is like a Stupid Marmot!!!"
```

A Random Insult Generator Demo Program: insult1.html

- Now let's try re-creating our random insult generator, using our function that picks random words. First, here's a reminder of what the code from Chapter 3 looked like:

A Random Insult Generator

- Notice that we end up repeating `words[Math.floor(Math.random() * length)]` quite a few times in this code. Using our `pickRandomWord` function, we could rewrite the program like this:

```
var randomBodyParts = ["Face", "Nose", "Hair"];
var randomAdjectives = ["Smelly", "Boring", "Stupid"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];

// Join all the random strings into a sentence:
var randomString = "Your " + pickRandomWord(randomBodyParts) + 
" is like a " + pickRandomWord(randomAdjectives) + 
" " + pickRandomWord(randomWords) + "!!!";

randomString;
"Your Nose is like a Smelly Marmot!!!"
```

A Random Insult Generator

- There are two changes here. First, we use the `pickRandomWord` function when we need a random word from an array, instead of using `words[Math.floor(Math.random()*length)]` each time. Also, instead of saving each random word in a variable before adding it to the final string, we're adding the return values from the function calls directly together to form the string. A call to a function can be treated as the value that the function returns. So really, all we're doing here is adding together strings.
- As you can see, this version of the program is a lot easier to read, and it was easier to write too, since we reused some code by using a function.

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼ var pickRandomWord = function (words) {
5     return words[Math.floor(Math.random() * words.length)];
6 };
7 var randomBodyParts = ["Face", "Nose", "Hair"];
8 var randomAdjectives = ["Smelly", "Boring", "Stupid"];
9 var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];
10 // Pick a random body part from the randomBodyParts array:
11 var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
12 // Pick a random adjective from the randomAdjectives array:
13 var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
14 // Pick a random word from the randomWords array:
15 var randomWord = randomWords[Math.floor(Math.random() * 5)];
16 // Join all the random strings into a sentence:
17 var randomString = "Your " + randomBodyPart + " is like a " +
18 randomAdjective + " " + randomWord + "!!!!";
19 document.write(randomString+"<br>");
20 </script>
21 </body>
22 </html>
```



Making the Random Insult Generator into a Function

PART 3

Making the Random Insult Generator into a Function

- We can take our random insult generator one step further by creating a larger function that produces random insults. Let's take a look:

```
generateRandomInsult = function () {  
  var randomBodyParts = ["Face", "Nose", "Hair"];  
  var randomAdjectives = ["Smelly", "Boring", "Stupid"];  
  var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];  
  // Join all the random strings into a sentence:  
  var randomString = "Your " + pickRandomWord(randomBodyParts) +  
    " is like a " + pickRandomWord(randomAdjectives) +  
    " " + pickRandomWord(randomWords) + "!!!";  
  
  ❶ return randomString;  
};
```

```
generateRandomInsult();  
"Your Face is like a Smelly Stick!!!"  
generateRandomInsult();  
"Your Hair is like a Boring Stick!!!"  
generateRandomInsult();  
"Your Face is like a Stupid Fly!!!"
```

Making the Random Insult Generator into a Function

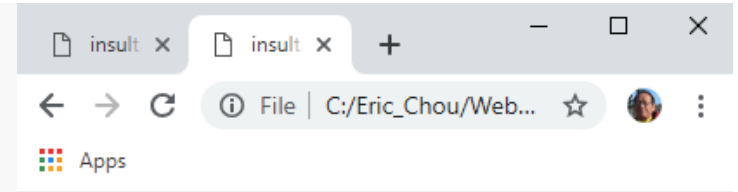
Making the Random Insult Generator into a Function

- Our new **generateRandomInsult** function is just the code from before placed inside a function with no arguments. The only addition is at ❶, where we have the function return **randomString** at the end. You can see a few sample runs of the preceding function, and it returns a new insult string each time.
- Having the code in one function means we can keep calling that function to get a random insult, instead of having to copy and paste the same code every time we want a new insult.



Demo Program: insult2.html

```
1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼ var pickRandomWord = function (words) {
5     return words[Math.floor(Math.random() * words.length)];
6 };
7
8 ▼ var generateRandomInsult = function () {
9     var randomBodyParts = ["Face", "Nose", "Hair"];
10    var randomAdjectives = ["Smelly", "Boring", "Stupid"];
11    var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];
12    // Join all the random strings into a sentence:
13    var randomString = "Your " + pickRandomWord(randomBodyParts) +
14                      " is like a " + pickRandomWord(randomAdjectives) +
15                      " " + pickRandomWord(randomWords) + "!!!";
16    return randomString;
17 };
18 ▼ for (var i=0; i<3; i++){
19     var randomString = generateRandomInsult();
20     document.write(randomString+"<br>");
21 }
22 </script>
23 </body>
24 </html>
```



Your Nose is like a Boring Stick!!!
Your Nose is like a Boring Monkey!!!
Your Face is like a Boring Rat!!!

Return as a Break Level

SECTION 18

Leaving a Function Early with return

- One common way to use return is to leave a function early if any of the arguments to the function are *invalid*; that is, if they're not the kind of arguments the function needs in order to run properly.
- For example, the following function returns a string telling you the fifth character of your name. If the name passed to the function has fewer than five characters, the function uses return to leave the function immediately.
- This means the return statement at the end, which tells you the fifth letter of your name, is never executed.



Return as a break level

```
var fifthLetter = function (name) {  
  ❶ if (name.length < 5) {  
    ❷ return;  
  }  
  
  return "The fifth letter of your name is " + name[4] + ".";  
};
```

- At ❶ we check to see whether the length of the input name is less than five. If it is, we use return at ❷ to exit the function early.



Return as a break level

- Let's try calling this function.

```
> fifthLetter("Nicholas");  
"The fifth letter of your name is o."
```
- The name *Nicholas* is longer than five characters, so **fifthLetter** completes and returns the fifth letter in the name *Nicholas*, which is the letter *o*. Let's try calling it again on a shorter name:

```
> fifthLetter("Nick");  
undefined
```
- When we call **fifthLetter** with the name *Nick*, the function knows that the name isn't long enough, so it exits early with the first return statement at ❷. Because there is no value specified after the return at ❷, the function returns **undefined**.

Replacement of else by return [Optional]

SECTION 19

Replacement of Else by Return

```
var medalForScore = function (score) {  
  ❶ if (score < 3) {  
    return "Bronze";  
  }  
  
  ❷ if (score < 7) {  
    return "Silver";  
  }  
  
  ❸ return "Gold";  
};
```

- We can use multiple return keywords inside different if statements in a function body to have a function return a different value depending on the input. For example, say you're writing a game that awards players medals based on their score. A score of 3 or below is a bronze medal, scores between 3 and 7 are silver, and anything above 7 is gold. You could use a function like **medalForScore** to evaluate a score and return the right kind of medal, as shown here:

Function as HTML Wrapper [Optional]

SECTION 20

HTML Wrapper Demo Program: wrapper.html

- In order to write an object to the document, functions can be used as HTML wrappers.
- Using function calls can make repetition more efficient.
- The template can be pre-defined and then, brought into JavaScript section.
- Template literals can be used (but not demonstrated in here.)
- The pre-defined template is in the field.html file.

```

1 ▼ <html>
2 ▼ <body>
3 ▼ <script>
4 ▼ function print(student){
5     return '<fieldset><ul>'+
6         '<li>Name: '+student.name+'</li>'+
7         '<li>Age: '+student.age+'</li>'+
8         '<li>Address: '+student.address+', '+student.city+', '+student.state+', '+
9         student.zipcode+'</li>'+
10        '</ul></fieldset>';
11 }
12 ▼ var s1 = {
13     name: "Tommy Jones",
14     age: 15,
15     address: "1 A Street",
16     city: "Los Angeles",
17     state: "CA",
18     zipcode: "90007"
19 };
20 ▼ var s2 = {
21     name: "Nancy Adams",
22     age: 16,
23     address: "100 B Street",
24     city: "Las Vegas",
25     state: "NV",
26     zipcode: "80109"
27 };
28 document.write(print(s1));
29 document.write(print(s2));
30 </script>
31 </body>
32 </html>

```

- Name: Tommy Jones
- Age: 15
- Address: 1 A Street, Los Angeles, CA, 90007

- Name: Nancy Adams
- Age: 16
- Address: 100 B Street, Las Vegas, NV, 80109