JavaScript

Javascript intro

- JavaScript is a client-side language designed to work in the browser
- Not restricted to browsers
- Tightly integrated with HTML
- JavaScript programs are executed by a JavaScript interpreter built into the browser.

Javascript intro (cont)

- When the browser requests such a page,
  - the server sends the full content of the document, including HTML and JavaScript statements
- When the page loads,
  - HTML content is rendered line by line until a JavaScript opening tag is read, at which time the JavaScript interpreter takes over
- When the closing JavaScript tag is reached,
  - the HTML processing continues

What JavaScript Is Not

JavaScript is not Java

Javascript

- developed at Netscape
- embedded in a Web page and run in a browser
- loosely typed and flexible
- variables, parameters, and function return types do not have to be declared
- interpreted by a JavaScript engine that lives in the browser

Java

- Developed at Sun Microsystems
- can be independent of a Web page
- Strongly typed language with strict guidelines
- Java data types must be declared
- Java programs are compiled

What JavaScript is Not

JavaScript is not HTML

- can be embedded in an HTML document
- contained within HTML tags
- own syntax rules and expects statements to be written in a certain way
- JavaScript doesn’t understand HTML
- but it can contain HTML content within its statements

What JavaScript is Not (storage and access)

- Not used to read or write the files on client machines with the exception of writing to cookies and local storages
- does not let you write to or store files on the server
- does not open or close windows already opened by other applications
- cannot read from an opened Web page that came from another server.

What JavaScript is Not (object orientation)

- JavaScript is object based but not strictly object oriented
- it does not support the traditional mechanism for inheritance and classes found in object-oriented programming languages, such as Java and C++
- The terms private, protected, and public do not apply to JavaScript methods as with Java and C++
- JavaScript is not the only language that can be embedded in an application.
- VBScript, for example, developed by Microsoft, is similar to JavaScript, but is embedded in Microsoft's Internet Explorer, Office and others.

What JavaScript Is Used For

- to detect and react to user-initiated events
  - mouse going over a link or graphic
- can improve a Web site with
  - navigational aids
  - scrolling messages and rollovers
  - dialog boxes
  - dynamic images
- lets you control the appearance of the page as the document is being parsed
- Without any network transmission, it lets you validate what the user has entered into
- a form before submitting the form to the server
- It can test to see if the user has plugins and send the user to another site to get the plug-ins if needed.

What JavaScript Is Used For

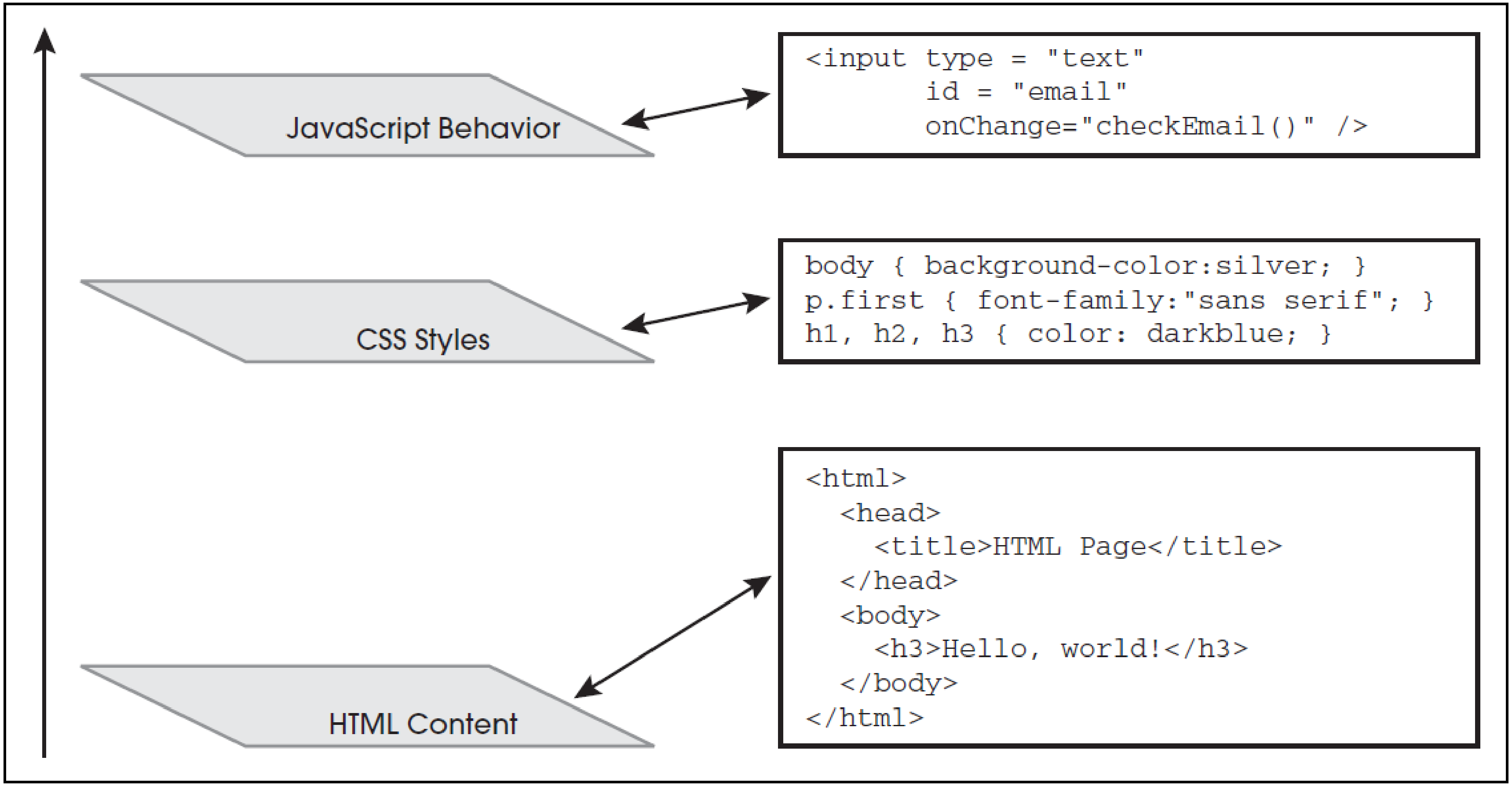
- It has string functions and supports regular expressions to check for
  - valid e-mail addresses
  - Social Security numbers
  - credit card data
- JavaScript serves as a programming language.
- Its core language describes such
  - basic constructs as variables and data types
  - control loops
  - *if/else statements*
  - *switch statements*
  - *Functions*
  - *Objects.*

What JavaScript Is Used For

used for

- arithmetic calculations
- manipulates the date and time
- works with arrays, strings, and objects
- It handles user-initiated events, sets timers
- changes content and style on the fly
- reads and writes cookie, localStorage, sessionStorage values
- dynamically creates HTML, Canvas graphics
- load content from server
- works as a background service
- HTML 5 enabled features

The Three Layers



JavaScript Behavior

```
<input type = "text"
       id = "email"
       onChange="checkEmail()" />
```

CSS Styles

```
body { background-color:silver; }
p.first { font-family:"sans serif"; }
h1, h2, h3 { color: darkblue; }
```

HTML Content

```
<html>
  <head>
    <title>HTML Page</title>
  </head>
  <body>
    <h3>Hello, world!</h3>
  </body>
</html>
```

Javascript and event (example)

```
<html>
  <head><title>Event</title></head>
<body>
  <form>
       <input type ="button“
       value = “Click me"
       onClick="alert(‘Clicked!')" />
  </form>
</body>
</html>
```

JavaScript and Events

onAbort	*Image loading was interrupted.*
onBlur	*The user moved away from a form element.*
onChange	*The user changed a value in a form element.*
onClick	*The user clicked a button-like form element.*
onError	*Error when loading an image.*
onFocus	*The user activated a form element.*
onLoad	*The document finished loading.*
onMouseOut	*The mouse moved away from an object.*
onMouseOver	*The mouse moved over an object.*
onSubmit	*The user submitted a form.*
onUnLoad	*The user left the window or frame.*

Standardizing JavaScript and the W3C

- 1990s **Internet Explorer** vs Netscape
  - new enhancements
  - proprietary features
  - incompatibilities , difficult to view a Web site the same way in the two browsers.
- **ECMAScript**
  - Netscape submitted JavaScript to ECMA International for Standardization.
  - European Computer Manufacturers Association (ECMA) organization that standardizes information
  - **ECMAScript** is a standard. While **JavaScript** is the most popular *implementation* of that standard. JavaScript implements ECMAScript and builds on top of it.

Javascript versions ES1 – ES5

- **ES** is simply short for **ECMAScript**. Every time you see **ES** followed by a number, it is referencing an edition of ECMAScript. In fact, there are 10 editions of ECMAScript published. Lets dive into them:
- ***ES1****: June 1997*
- ***ES2****: June 1998*
- ***ES3****: Dec. 1999*
- ***ES4****:Abandoned*
- **ES5***:December 2009:* Nearly 10 years later. It would then take almost six years for the next version of **ECMAScript** to be released.

Javascript versions ES2015 - **ES**2019

- **ES6 / ES2015** *June 2015*: **ES6 and ES2015 are the same thing**. The committee that oversees ECMAScript specifications made the decision to move to annual updates. With this change, the edition was renamed to **ES 2015** to reflect the year of release**.**
- **ES2016 (ES7)** *June 2016*:
- **ES2017 (ES8)** *June 2017*:
- **ES2018 (ES7)** *June 2018*:
- **ES2019 (ES8)** *June 2019*:

DOM support

- After ECMAScript was released, W3C began work on a standardized DOM, known as DOM Level 1, and recommended in late 1998.
- DOM Level 2 was published in late 2000.
- In fact 95% of all modern browsers support the DOM specifications.
- latest ECMAScript *http://www.ecmascript.org/.*

JavaScript Objects

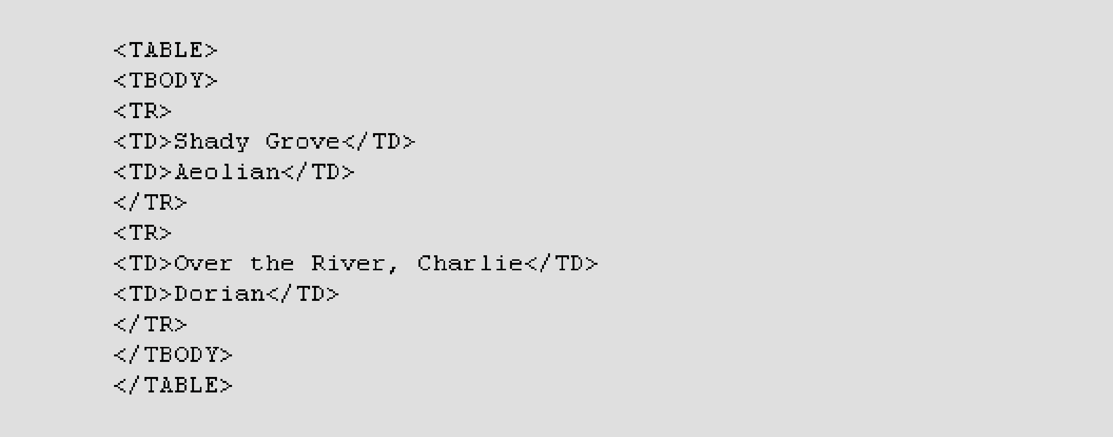
- Everything you do in JavaScript involves objects.
- JavaScript sees a Web page as many different objects,
  - the browser object
  - the document object
  - each element of the document as an object
    - Forms, images, and links are also objects.
  - In fact every HTML element in the page can be viewed as an object.
    - H1, P,TD, FORM, and HREF elements
- JavaScript has a set of its own core objects
  - strings, numbers, functions, dates, and so on
- JavaScript allows you to create your own objects.

- ***document.***write("Hello, world");
- ***object***.**method()**
  - ***Object is*** current page
  - **method** is **write**

The Document Object Model

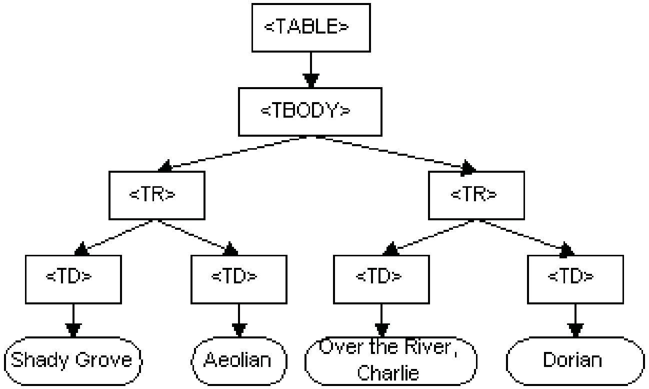
- Browser stores interpretation of the HTML page as a **model**, called the Document Object Model.
- The model is similar to the structure of a family tree
  - consisting of parents
  - children
  - Siblings
- These elements are referred to as **nodes**, with the **root** parent node of the tree at the top

Example DOM



```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

A graphical representation of the DOM of the example table is:



graphical representation of the DOM of the example table

DOM provides

- With this upside down tree model every element of the document becomes an object accessible by JavaScript (and other applications)
- Ability to control over an entire Web page
  - Navigate
  - Create
  - Add
  - Modify
  - delete the elements
  - and their content dynamically.

Where to Put JavaScript

- **Embedded**
  - *<head> and </head>*
    - developers prefer
    - best place to store function definitions and objects
  - *<body> and </body>*
    - text displayed at a specific spot in the document, you might want to place the JavaScript
  - *<head> and </head>* + *<body> and </body>*

- **External**
  - sharable by multiple pages
  - separate the HTML/CSS content from the programming logic

External JavaScript (example)

- // The external file called "welcome.js"

  ```
  function myFunction() {          alert(“Hello");          }
  ```

-
  ```
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN“
  "http://www.w3.org/TR/html4/strict.dtd">
  ```

```
<html>
   <head><title>External File</title>
          <script type="text/javascript" src="welcome.js"></script>
   </head>
   <body bgColor="lavender">
          <input type="button" onClick=" myFunction()"
   value=“Click!" />
   </body>
</html>
```

JavaScript fundamentals

- Unlike Perl and Python scripts, JavaScript scripts are not stand-alone programs on front-end side.
- They are run in the context of an HTML document
- When programming on the client side,
  - create an HTML document in your favorite text editor
  - *the file you create is an HTML document,* its name must include either an *.html or .htm extension*
  - execute it in your browser window

Syntactical Details

Case Sensitivity

JavaScript names are **case sensitive**

- Variables, keywords, objects, functions
- Boolean value *true with* any uppercase letters (e.g., TrUE), JavaScript will not recognize it and will produce an error or simply ignore the JavaScript code.
- Although most names favor lowercase, some JavaScript names use a combination of upper and lowercase (e.g., *onClick, Math.floor, Date.getFullYear)*.

Syntactical Details
Free Form

- JavaScript ignores whitespace appears between words.
  - spaces, tabs, newlines
- **Same statement**

```
1.  var name="Tom";

2.  var          name  =
                 "Tom";
```

- **Not same statement**
  - onMouseOver()
  - on Mouse Over()

Syntactical Details

Free Form

Whitespace is preserved when it is embedded within a string or regular expression.

- ***“Hello   there”*** *will be preserved because* it is enclosed within double quotes.

- Because extra whitespace is ignored by the JavaScript interpreter
  - you are free to indent
  - break lines
  - organize easier to read and debug.

Syntactical Details

Reserved words (keywords)

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	delete
do	double	else	extends	false	final
finally	float	for	function	goto	if
implements	import	in	instanceof	int	interface
long	native	new	null	package	private
protected	public	return	short	static	super
switch	synchronized	this	throw	throws	transient
true	try	typeof	var	void	volatile
while	with				

Syntactical Details
Statements and Semicolons

- var name = "Ellie" <– *no semicolon, valid*
- var name = "Ellie"**; <– *better***

incorrect:

- var name = "Ellie" document.write("Hi "+name);
  - *Because, two statements*

correct:

- var name = "Ellie"**; document.write("Hi " + name);**
  - ***semicolon needed*** *to separate two statements on the same line*

Syntactical Details
Comments

Single-line comments start with a **double slash**:

// This is a comment

Block of comments, use the **/* */** symbols:

/ This is a block of comments*

that continues for a number of lines

…

*/

Generating HTML and Printing Output Strings

- String quote
  - “ this is a string “
  - ‘ this is a string ‘
  - ` this is a string ` **//backtick**
- quotes can hide quotes;
  - “ I don't care “
  - ‘ He cried, “Ahoy!“ ‘
  - ‘ He cried, `Ahoy!` ‘

Generating HTML and Printing Output
Strings concatenation

- Concatenation is caused when two strings are joined together.
- The plus (+) sign to concatenate strings;
- "hot" + "dog" or "San Diego" + "<br />“
- When you write something inside ${} in a template literal, its result will be
  - computed,
  - converted to a string,
  - and included at that position.
- *console.log(`half of 100 is ${ 100 / 2 }`)*
- The example produces “half of 100 is 50”.

write() and writeln() – but be aware!

- One of the most important features of client-side JavaScript is its ability to generate pages dynamically
- Data, text, and HTML itself can be written to the browser on the fly
- *write(), writeln() methods are a special kind of built-in function used to output* HTML to the document as it is being parsed. Use writeln() inside <pre> tag
- *put the statement in the body of the document at the* place where you want the text to appear when the page is loaded. *(rather than in the header)*

write() and writeln() example

```
<html>
<head><title>Printing Output</title></head>
<body>
<script type="text/javascript">
 document.write(“no newline ");
 document.writeln(“newline1");
 document.write(“newline 2<br />newline3");
</script>
```

Avoid document.write, specifically for scripts injection

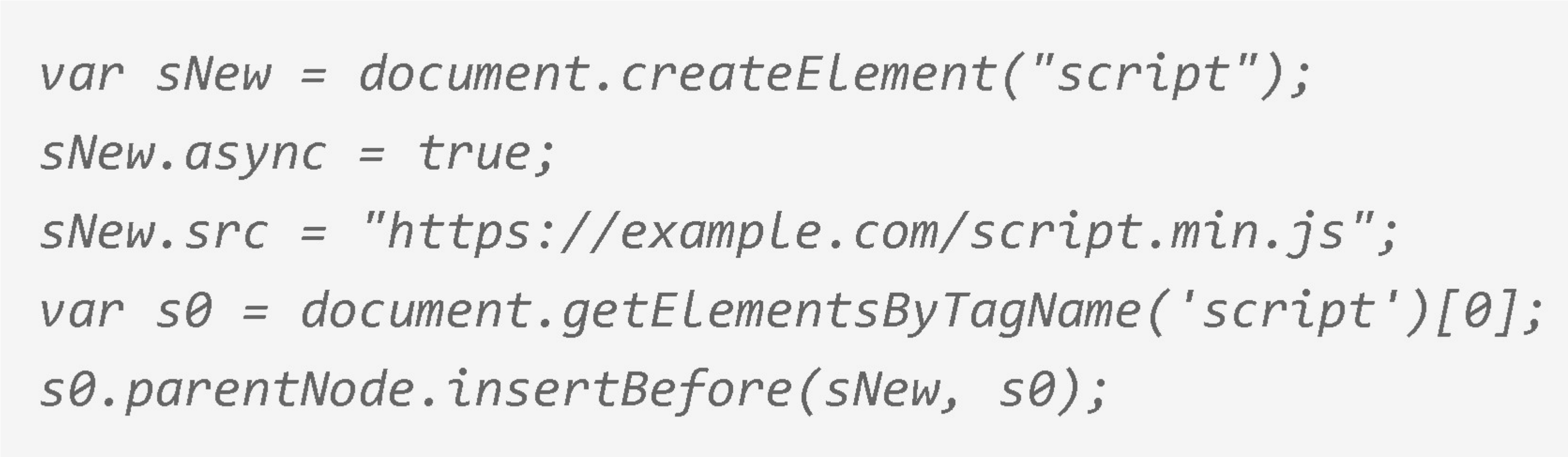
- When you use the following Javascript instruction to inject a script:

```
document.write('<script src="https://example.com/script.js"></script>’);
```

- The web browser has to pause the HTML parsing, it is forced to wait for the resource to load and to be executed.
- The situation could even be more harmful, as the browser will also have to wait for additional scripts that could be injected subsequently!

Avoid document.write

- If the DOM tree has already been built, the use of document.write will force the browser to build it again /not only script injection/
- When document.write writes to the document stream – document will reset the current document then build again



```
var sNew = document.createElement("script");
sNew.async = true;
sNew.src = "https://example.com/script.min.js";
var s0 = document.getElementsByTagName('script')[0];
s0.parentNode.insertBefore(sNew, s0);
```

The Building Blocks: Data Types, Literals, and Variables

primitive and composite

Primitive Data Types

Primitive data types are the simplest building blocks of a program

- numeric
- String
- Boolean
- two special types that consist of a single value:
  - null
  - undefined

Data Types

- **Numeric Literals**

12345	integer
23.45	float
.234E-2	scientific notation
.234e+3	scientific notation
0x456fff	hexadecimal
0x456FFF	hexadecimal
0777	octal

Data Types

- **String Literals and Quoting**
  - “hello”
- An empty set of quotes is called the null string
- *“5” is a string*
- *5 is a number*

Escape Sequence

\'	Single quotation mark
\"	Double quotation mark
\t	Tab
\n	Newline
\r	Return
\f	Form feed
\b	Backspace
\e	Escape
\\	Backslash

Special Escape Sequences

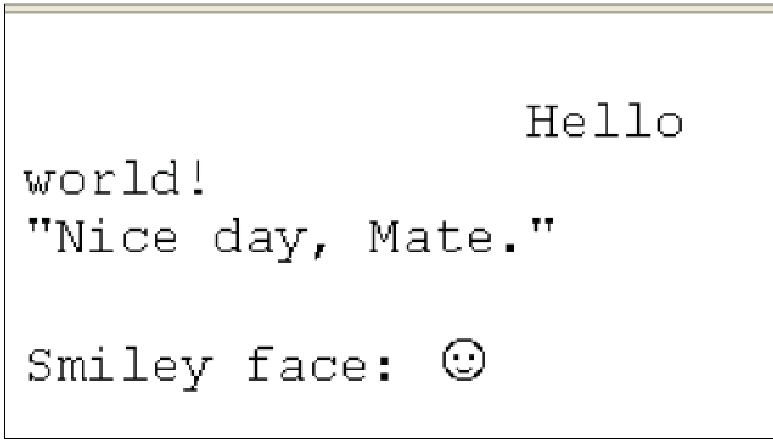
- *\XXX The character with the Latin-1 encoding specified by up to* three octal digits XXX between 0 and 377.
    - \251 is the octal sequence for the copyright symbol.

- *\xXX The character with the Latin-1 encoding specified by the* two hexadecimal digits XX between 00 and FF.
    - \xA9 is the hexadecimal sequence for the copyright symbol.

- *\uXXXX The Unicode character specified by the four hexadecimal* digits XXXX.
    - \u00A9 is the Unicode sequence for the copyright symbol.

Escape Sequence example

```
<body>
<pre>
    <script type="text/javascript">
    <!-- Hide script from old browsers.

    document.write("\t\tHello\nworld!\n");

    document.writeln("\"Nice day, Mate.\"\n");

    document.writeln('Smiley face:<font size="+3"> \u263a\n');

    //End hiding here. -->
    </script>
</pre>
</body>
```



```
                         Hello
world!
"Nice day, Mate."

Smiley face:  ☺
```

Putting strings together

- "pop" + "corn“ results in popcorn
- "Route " + 66 results in Route66
- *5 + 100 results in 105*
- *“5”+ 100 results in “5100”.*

Boolean Literals.

- **Boolean literals are logical values that have only one of two values,**
  - *true*
  - *False*
- *You can think of the values as yes or no, on or off, or 1 or 0.*
- *They are used* to test whether a condition is true or false.
- When using numeric comparison and equality operators, the value *true evaluates to 1 and false evaluates to 0.*

- answer1 = **true;**
- if (answer2 == **false) { do something; }**

Variables

- Languages that require that you specify a data type are called “strongly typed” languages.
- JavaScript, conversely, is a dynamically or loosely typed language, meaning that you do not have to specify the data type of a variable.
- num = 5; *// name is "num", value is 5, type is numeric*
- friend = "Peter"; *// name is "friend", value is "Peter“, type is string*

Variable Declaration vs Initialization

- var hello;

  console.log(hello); // prints out ‘undefined’

In JavaScript, variables are initialized with the value of undefined when they are created.

- var declaration=“Hello”;

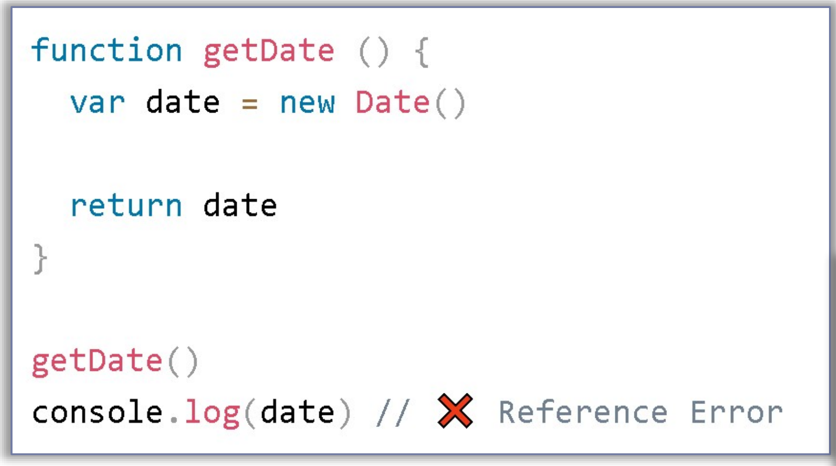
  console.log(declaration); // prints out ‘Hello’

Scope

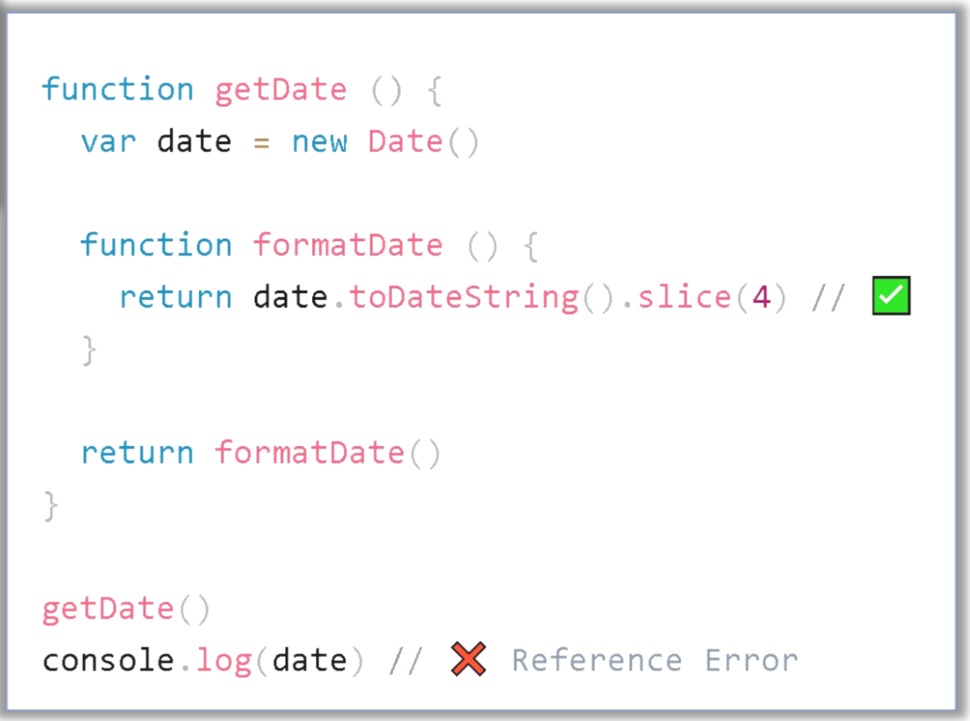
- Scope defines where variables and functions are accessible inside of your program.
- In JavaScript, there are two kinds of scope - **global scope**, and **function scope**. According to the official spec,

> *"If the variable statement occurs inside a FunctionDeclaration, the variables are defined with function-local scope in that function."*

Scope example

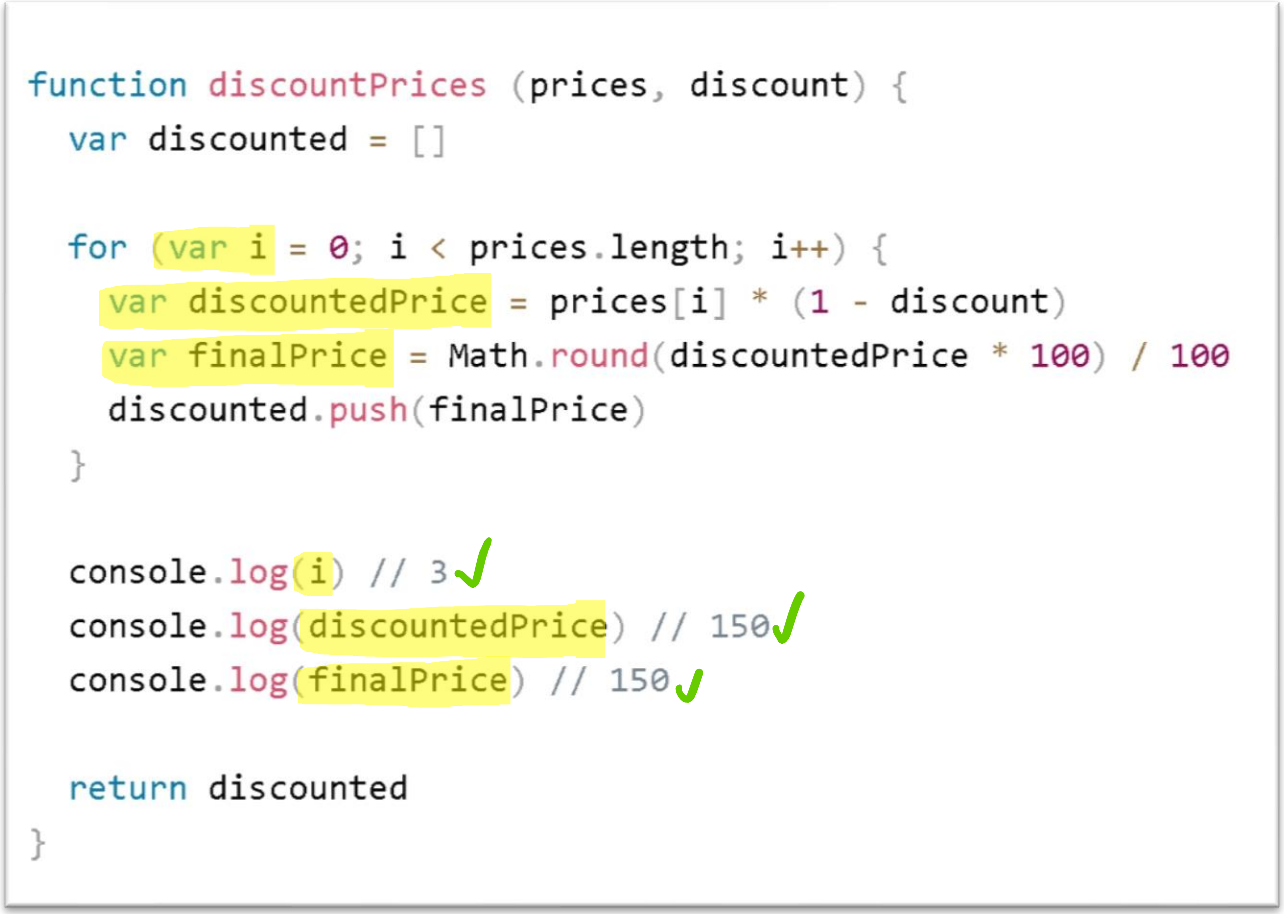


```
function getDate () {
  var date = new Date()

  return date
}

getDate()
console.log(date) // ❌ Reference Error
```



```
function getDate () {
  var date = new Date()

  function formatDate () {
    return date.toDateString().slice(4) // ✅
  }

  return formatDate()
}

getDate()
console.log(date) // ❌ Reference Error
```

var Scope example

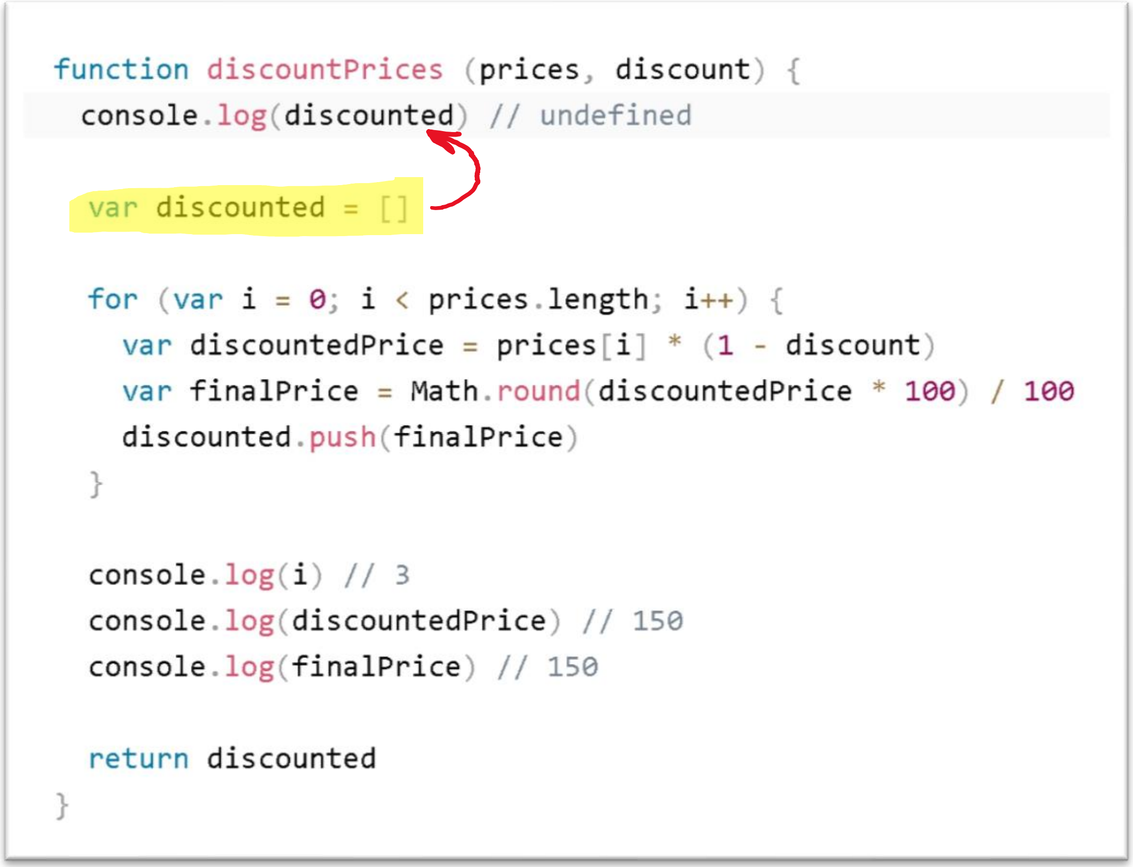


```
function discountPrices (prices, discount) {
  var discounted = []

  for (var i = 0; i < prices.length; i++) {
    var discountedPrice = prices[i] * (1 - discount)
    var finalPrice = Math.round(discountedPrice * 100) / 100
    discounted.push(finalPrice)
  }

  console.log(i) // 3 ✓
  console.log(discountedPrice) // 150 ✓
  console.log(finalPrice) // 150 ✓

  return discounted
}
```

Hoisting

before it was actually declared, you'll just get undefined.

In our mind variables can't be used before declaration!



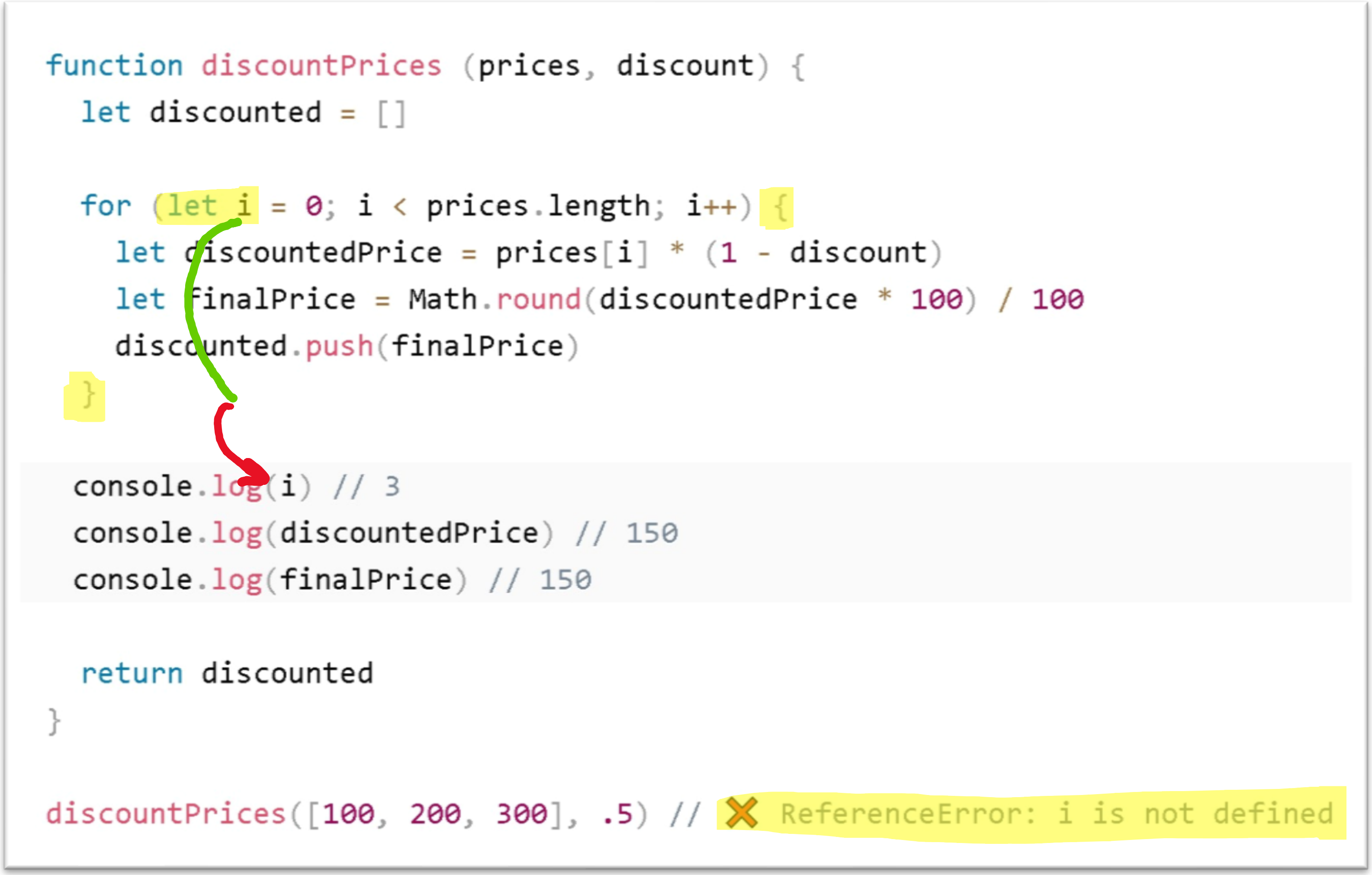
```
function discountPrices (prices, discount) {
  console.log(discounted) // undefined

  var discounted = []

  for (var i = 0; i < prices.length; i++) {
    var discountedPrice = prices[i] * (1 - discount)
    var finalPrice = Math.round(discountedPrice * 100) / 100
    discounted.push(finalPrice)
  }

  console.log(i) // 3
  console.log(discountedPrice) // 150
  console.log(finalPrice) // 150

  return discounted
}
```

var VS let

- **var** - function scoped
- **let** - block scoped, as well as any nested blocks.
  - block means curly brace { }

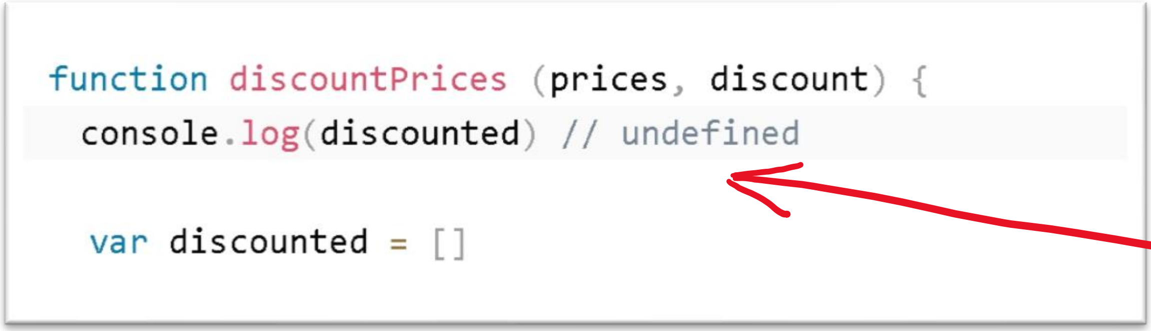
- let keyword is available inside the “block” that it was created in as well as any nested blocks. When I say “block”, I mean anything surrounded by a curly brace {} like in a for loop or an if statement.

let Example:

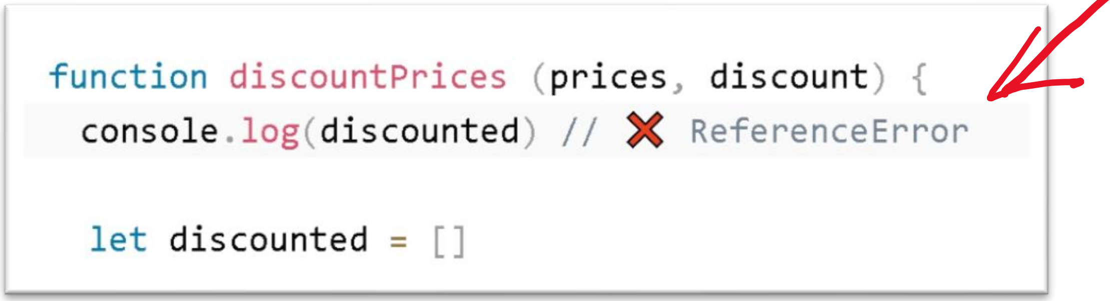


```
function discountPrices (prices, discount) {
  let discounted = []

  for (let i = 0; i < prices.length; i++) {
    let discountedPrice = prices[i] * (1 - discount)
    let finalPrice = Math.round(discountedPrice * 100) / 100
    discounted.push(finalPrice)
  }

  console.log(i) // 3
  console.log(discountedPrice) // 150
  console.log(finalPrice) // 150

  return discounted
}

discountPrices([100, 200, 300], .5) // ✗ ReferenceError: i is not defined
```

Undefined or Reference error ?



```
function discountPrices (prices, discount) {
  console.log(discounted) // undefined

   var discounted = []
```

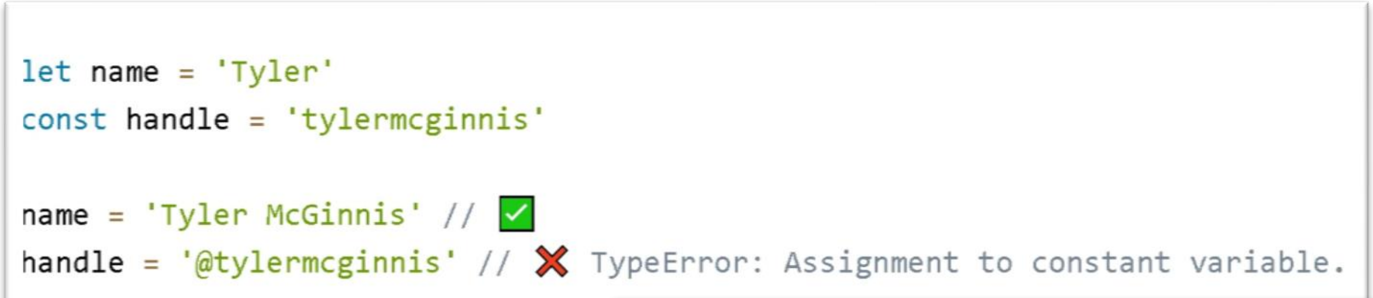


```
function discountPrices (prices, discount) {
  console.log(discounted) // ❌ ReferenceError

   let discounted = []
```

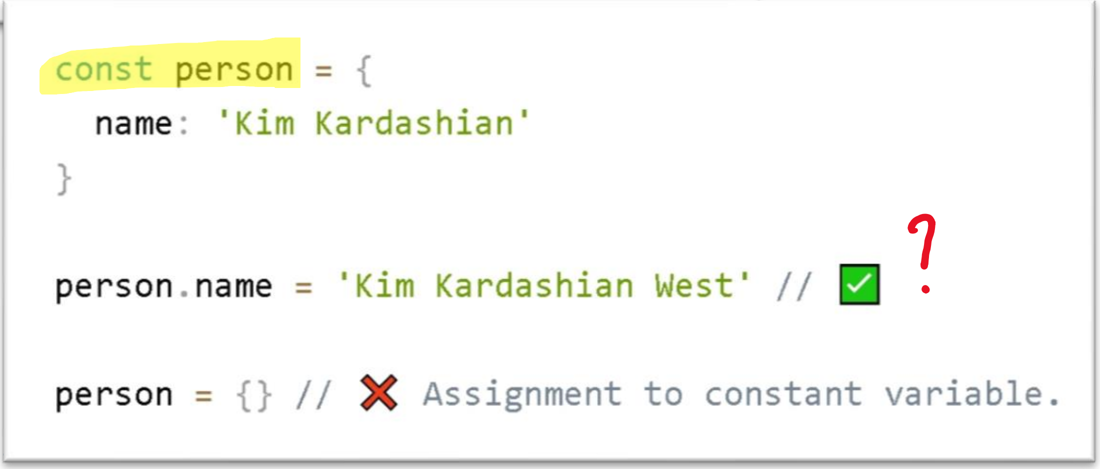
?

let VS const

- const is almost exactly the same as let.
- Once assigned a value **can’t reassign** (it doesn’t mean you can’t change the value) it to a new value.

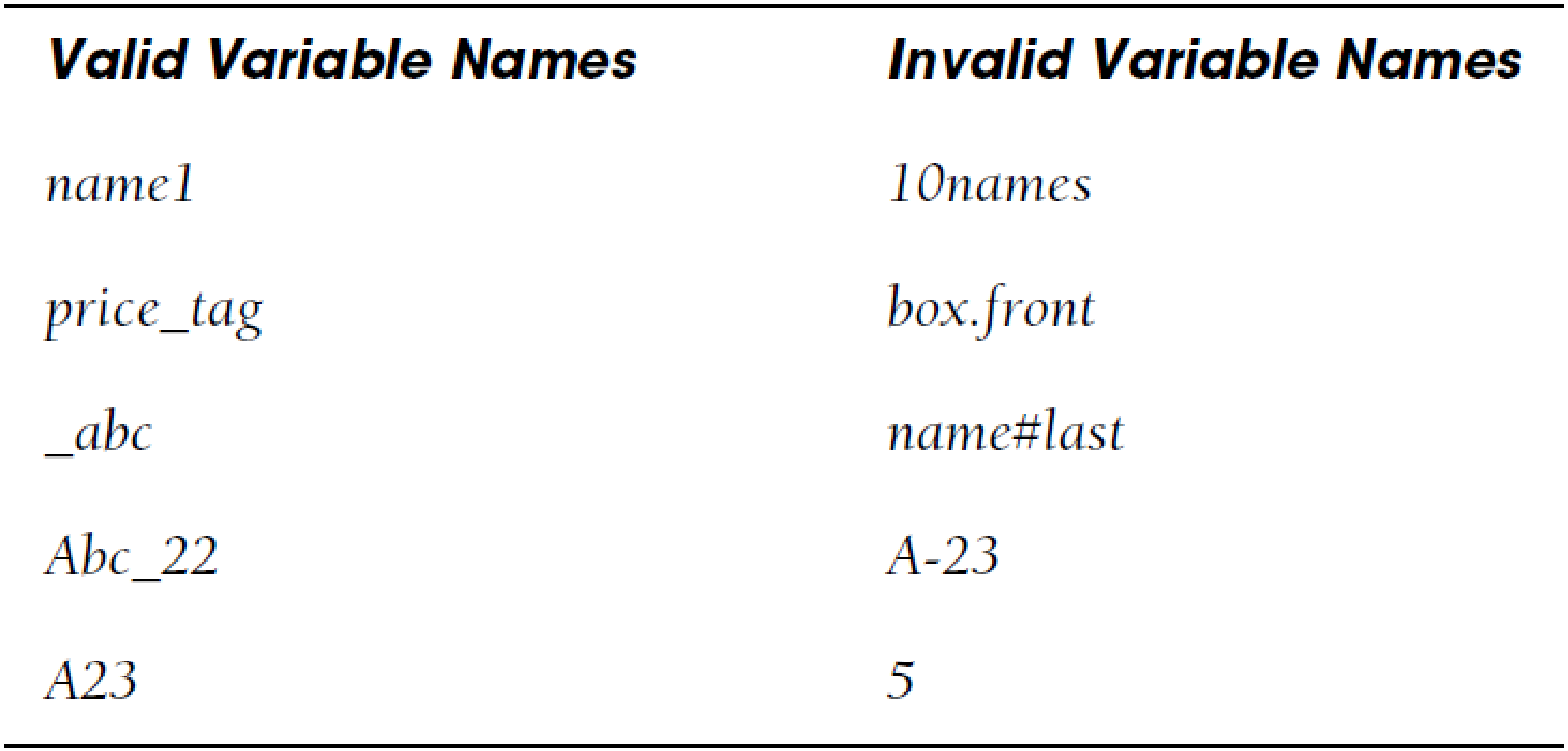


```
let name = 'Tyler'
const handle = 'tylermcginnis'

name = 'Tyler McGinnis' // ✓
handle = '@tylermcginnis' // ✗ TypeError: Assignment to constant variable.
```



```
const person = {
  name: 'Kim Kardashian'
}

person.name = 'Kim Kardashian West' // ✓ ?

person = {} // ✗ Assignment to constant variable.
```

Valid and invalid names



Valid Variable Names	Invalid Variable Names
name1	10names
price_tag	box.front
_abc	name#last
Abc_22	A-23
A23	5

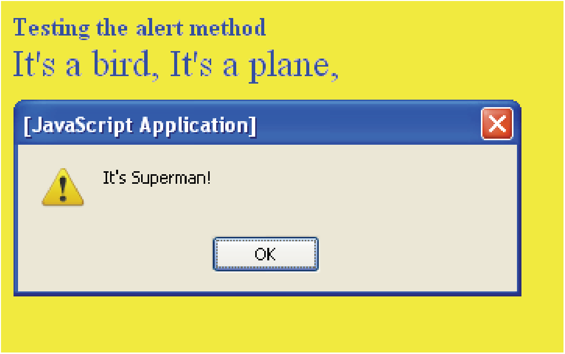
- var variable_name = value; // *initialized*
- var variable_name; // *uninitialized*
- variable_name; // *wrong*

- *var first_name="Ellie“*
- first_name ="Ellie";
- var first_name;

Dialog Boxes

Interacting with the User

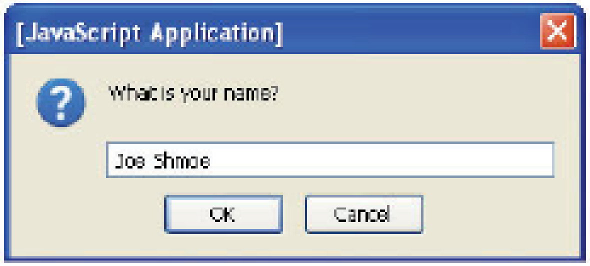
- *alert()*
- *prompt()*
- *confirm()*



Testing the alert method

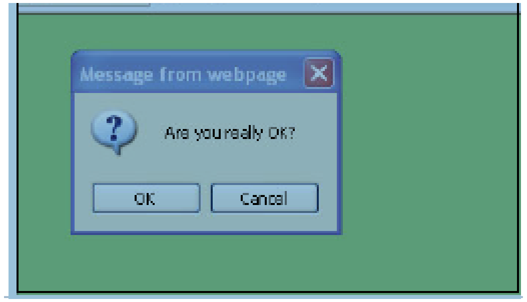
It's a bird, It's a plane,

It's Superman!



What is your name?

Joe Shmoe



Are you really OK?

Example

- **alert("Welcome to my world! " + name);**

- let age=**prompt("Tell me your age.", "Your age: ");**

- if(**confirm("Are you really OK?")** == true)
      alert("Then we can proceed!");
  else
      alert("We'll try when you feel better? ");

- What is wrong with the following alert box?
  - alert("Hello<br />", "world!<br />");

Operators

The operators on the same line are of equal precedence. The rows are in order of highest to lowest precedence.

Operator	***Description***	***Associativity***
()	*Parentheses*	*Left to right*
++ – –	*Auto increment, decrement*	*Right to left*
!	*Logical NOT*	*Right to left*
* / %	*Multiply, divide, modulus*	*Left to right*
+ –	*Add, subtract*	*Left to right*
+	*Concatenation*	*Left to right*
< <=	*Less than, less than or equal to*	*Left to right*
> >=	*(greater than) or equal to*	*Left to right*
= = ,!=	*Equal to, not equal to*	*Left to right*
= = = ,!= =	*Identical to (same type), not identical to*	*Left to right*

Operators

<table>
<tr><th>Operator</th><th>Description</th><th>Associativity</th></tr>
<tr><td>&</td><td><em>Bitwise AND</em></td><td><em>Left to right</em></td></tr>
<tr><td>|, ^, ~</td><td><em>Bitwise OR, XOR, NOT</em></td><td></td></tr>
<tr><td><<</td><td><em>Bitwise left shift</em></td><td></td></tr>
<tr><td>>></td><td><em>Bitwise right shift</em></td><td></td></tr>
<tr><td>>>></td><td><em>Bitwise zero-filled, right shift</em></td><td></td></tr>
<tr><td>&&</td><td><em>Logical AND</em></td><td><em>Left to right</em></td></tr>
<tr><td>||</td><td><em>Logical OR</em></td><td><em>Left to right</em></td></tr>
<tr><td>? :</td><td><em>Ternary, conditional</em></td><td><em>Right to left</em></td></tr>
<tr><td>= += – = *= /= %= <<= >>=</td><td><em>Assignment</em></td><td><em>Right to left</em></td></tr>
<tr><td>,</td><td><em>(comma)</em></td><td></td></tr>
</table>

Example

- let result = 5 + 4 * 12 / 4;

  could be written
- result = (5 + ((4 * 12) / 4));

Arithmetic Operators

Operator/Operands	***Function***
x + y	*Addition*
x – y	*Subtraction*
*x * y*	*Multiplication*
x / y	*Division* (1/2 returns 0.5)
x % y	*Modulus*

Shortcut Assignment Operators

Operator	***Example***	***Meaning***
=	*var x = 5;*	*Assign 5 to variable x.*
+=	*x += 3;*	*Add 3 to x and assign result to x.*
–=	*x –= 2;*	*Subtract 2 from x and assign result to x.*
***=**	*x *= 4;*	*Multiply x by 4 and assign result to x.*
/=	*x /= 2;*	*Divide x by 2 and assign result to x.*
%=	*x %= 2;*	*Divide x by 2 and assign remainder to x.*

Operator	**Function**	**What It Does**
++*x*	*Pre-increment*	*Adds 1 to x*
x++	*Post-increment*	*Adds 1 to x*
– –*x*	*Pre-decrement*	*Subtracts 1 from x*
x– –	*Post-decrement*	*Subtracts 1 from x*

Concatenation Operator

Operator	***Example***	***Meaning***
+	*"hot" + "dog"*	*Joins two strings; creates* **"hotdog"**.
	"22" + 8	*Converts number 8 to string "8", then concatenates* resulting in ***"228"***. *In statements involving other* operators, JavaScript does not convert numeric values to strings.

+=	*x* ="*cow*";
	x += "*boy*";
		x becomes "cowboy"

Comparison Operators

Operator/Operands	**Function**
x == y	*x is equal to y*
x != y	*x is not equal to y*
x > y	*x is greater than y*
x >= y	*x is greater than or equal to y*
x < y	*x is less than y*
x <= y	*x is less than or equal to y*
x = = = y	*x is identical to y in value and type*
x != = y	*x is not identical to y*

Example

Test	***Are They Equal?***
"William" == "William"	*true*
"william" == "William"	*false*
5 == 5.0	*true*
"54" == 54	*true*
"5.4" == 5.4	*true*
*NaN == NaN (*Not a Number*)*	*false*
null == null	*true*
–0 == +0	*true*
false == false	*true*
true == 1	*true*
null == undefined	*true*

Comparing Strings

Example How Operator Compares Strings

- ***"string1" > "string2"*** *"string1" is greater than "string2“*
- ***"string1" >= "string2"*** *"string1" is greater than or equal to "string2“*
- ***"string1" < "string2"*** *"string1" is less than "string2“*
- ***"string1" <= "string2"*** *"string1" is less than or equal to "string2"*

Logical AND Examples

Expression	**What It Evaluates To**
true && false	*false*
true && true	*true*
"honest" && true	*true*
true && ""	*(empty string)*
true && "honest"	*honest*
5 && 0	*0*
5 && –6	*–6*
5 && false	*false*
null && 0	*null*
null && ""	*null*
null && false	*null*
"hello" && true && 50	*50*
"this" && "that"	*that*

- x ? y : z If x evaluates to true, the value of the expression becomes y, else the value of the expression becomes z

- **big = (x > y) ? x : y If *x is greater than y, x is assigned to*** variable *big, else y is assigned to* variable *big*

- An *if/else statement instead of the conditional statement:*

- if (x > y) {
- big = x;
- }
- else{
- big = y;
- }