**JavaScript**: a programming language that adds complex interactive features to a website

•Developed to handle customer data validation for online commerce

•**World Wide Web** (**web**) allows data sharing across a network of linked documents

•Introducing scripting languages

•JavaScript is a type of programming language, specifically a scripting language

•**Programming language**: a set of instructions directing the actions of the computer

•Typically must be **compiled**into **machine code** by a program called a **compiler**

•**Scripting languages**: subcategory of programming languages that run directly from a program or **script**(are not compiled)

•Must be interpreted (read and scanned line-by-line) by an interpreter.

Introducing scripting languages (continued)

•**Markup languages**: languages that define the content, structure, and appearance of a document

•**HTML (Hypertext Markup Language)** defines the content and structure of a web page

•**CSS (Cascading Style Sheets)** defines how a web page will appear on a specified device

JavaScript and ECMAScript

•Developers struggled to reconcile JavaScript and **JScript**in the late 1990s

•JavaScript was submitted as a proposal for a standardized scripting language to the **European Computer Manufacturers Association (ECMA)** in 1997

•An ECMA committee developed a set of scripting language standards

•The specification for this scripting language, called **ECMAScript** (**ECMA-262**), is updated to a new edition annually

The DOM and the BOM

•Three foundations of the full implementation of JavaScript:

•ECMAScript (core of the language: syntax, keywords, properties, methods, etc.)

•**Document Object Model (DOM)**: describes how to access the contents of a web page and user actions within that page

•**Browser Object Model (BOM)**: describes how to access the features and behaviors of the browser itself

•DOM and BOM are **Application Programming Interfaces (APIs)**

•DOM specifications are managed by the **World Wide Web Consortium (W3C)**

•Current version is DOM Level 4, described as an ongoing “living standard”

•BOM is implemented by each browser application (largely the same among them)

Traditionally, a **two-tier system** consisting of:

•**Server**(**back end**): a device or application from which a client requests information; responsible for data storage, management, communication with external services, and heavy processing

•**Client**(**front end**): a device or application that presents an interface to the user and requests information from a server; responsible for gathering information from the user, submitting it to a server, then receiving, formatting, and presenting the results, and sometimes some related data processing

•Web’s two-tier client/server system consists of a web browser and web server that communicate via **Hypertext Transfer Protocol (HTTP)**

Adding databases and other applications to a web server yields a **three-tier client/server system** (a.k.a. **multitier client/server system**, ***n*-tier client/server system**): client tier, **processing tier** (a.k.a. **middle tier**), and data storage tier

JavaScript and client-side scripting

•HTML produces **static**documents, whereas **client-side scripting** provided by JavaScript can respond dynamically to user actions because it runs on a local browser (client)

•Security concerns limit where JavaScript can be used, what files it can access/ commands it can run on the client’s system, and its direct interactions with web servers operating at the processing tier

**Server-side scripting:** programming using a scripting language (e.g., PHP, ASP.NET, Python, Ruby) that is executed from a web server

•These languages work in the processing tier and can handle communication between the client and data storage tiers, often preparing and processing data

•JavaScript and server-side scripts operate in separate environments but must work together to deliver interactive websites to users

Allow the client to handle the user interface processing and other light processing such as data validation

•Allow the web server to perform intensive calculations and data storage

•Performing some processing on the client is beneficial because:

•Processing power is not limited to the capabilities of the server when processing is distributed among multiple client devices using a web application

•Transfer times across the Internet are minimized, increasing speed

•Server resource requirements (infrastructure, power use) are decreased, reducing costs

Use an **Integrated Development Environment (IDE)** to manage all of the facets of website development

•Use a **code editor** to manage coding in HTML, CSS, and JavaScript within a graphical interface

Syntax for embedding JavaScript code in a web page’s HTML file:<script>*statements*</script>

•Browser stops loading the page and processes the statements when it encounters an embedded script

A JavaScript program is composed of **statements**(lines of code)

•Sample statement showing optional (but desirable per convention) ending semicolon:document.write("<p>Plant choices</>");

JavaScript is considered an object-based programming language

•**Object**: programming code (including methods) and data (properties) that can be treated as an individual unit or component

•**Procedure**: logical unit of a computer program consisting of a group of statements that perform a specific task

•**Method**: a procedure associated with an object

•Sample method call on carLoan

object: carLoan.calcPayments(60)

•Methods often require that you provide data (**argument**[s]) within the parentheses, which is known as **passing arguments**

•**Property**: a piece of data associated with an object

•Sample statement assigning a value for the interestproperty of the carLoanobject: carLoan.interest= .0349;

DOM's Documentobject represents the entire content of the web page

•The Documentobject's write()method writes new content to a web page while it is being loaded

•Performs essentially the same function that you perform when you manually add text to the body of a standard web document

•Useful for incorporating constantly-changing data at load time

•Requires a **text string** (a.k.a. **literal string**), which is text contained in double or single quotation marks, as an argument

•Will overwrite the entire web page if used after the browser finishes loading the page

Case sensitivity in JavaScript

•Object names must always be all lowercase

•Incorrect capitalization, e.g. Document.write("Plant choices");or document.WRITE("Plant choices");, will cause errors

•Adding comments to a JavaScript program

•**Comments**: lines of code that are not processed by browsers, which you can use to add notes about your code

•**Line comment** syntax:let apple = "Fuji"; // Variable assignment

•**Block comment** syntax:

/\*

Here comes a function!

\*/

**Variable**: specific location in the computer's memory where a program stores a value

•First created and assigned a name, then used to store a value

Rules and conventions for variable names, known as **identifiers**:

•Must begin with an uppercase or lowercase ASCII letter, dollar sign, or underscore

•Can include numbers but not as the first character

•Cannot include spaces

•Cannot be a **reserved word** (a.k.a. **keyword**)—a special word that is part of JavaScript syntax

Best practice is to use **camel case**, e.g., myVariableName

**Declaring** a variable creates it and thus is mandatory prior to using the variable

•**Initializing**a variable assigns it an initial value and is optional

•Syntax for declaring and initializing a variable using the **assignment operator**:let *variable*= *value*;var *variable*= *value*;

•Syntax for declaring a variable with a constant (unchangeable) value: const *variable*= *value*;

•Sample statements that declare and/or initialize variables:

let salesTotal; // Declares

onlylet curOrder= 47.58; // Declares and initializes

salesTotal= curOrder;let orderNumber= "R0218", salesTotal= 47.58, curOrder;

**Expression**: a literal value or variable, or a combination of literal values, variables, operators, and other expressions, that can be evaluated by a JavaScript interpreter to produce a result

•Expressions are written using:

•**Operands**: variables and **literals**(values such as text strings or numbers)

•**Operators**: symbols used to manipulate operands

•Example uses of the addition (+) and assignment (=) operators:let salesTotal= 47.58, shippingCost= 10;let totalCost= salesTotal+ shippingCost;document.write("<p> Your total costs is $" + totalCost+ "</p>");Generates the HTML code <p>Your total cost is $57.58</p>5 + "2"returns the text string "52"

Expressions can be used to assign new values to variables at any point in a script

•Applies to variables declared with let or var; variables declared with const cannot be modified

•Variables need be declared only once, then can be assigned repeatedly

•Example:let totalSales= 0;let item1Sales = 50, item2Sales = 75, item3Sales = 40;totalSales= item1Sales + item2Sales + item3Sales;document.write("<p>Total sales = $" + totalSales+ "</p>");Writes the HTML <p>Total sales = $165</p>to the web page

**Event**: a specific circumstance (such as an action performed by a user or an action performed by the browser) that is monitored by JavaScript and that your script can respond to in some way

Not all events happen with all devices; here are a few examples (part of Figure 1-14)

Events are associated with HTML elements, with each element having its own set of available events

•The clickevent is available for many elements, including the aelement and form controls created with the inputelement

•The loadand unloadevents are available for the bodyelement

•**Event handler**: code that is executed in response to a specific event occurring on a specific element

•Syntax for an event handler included as an attribute of the initiating element:<*element*on*event*="*JavaScript code*">

•Example using the window.alert()method, which displays the string passed to it in a dialog box with an OK button:<input type="submit" onclick="window.alert

('Thanks for your

In JavaScript, you look up an element by its idvalue using the getElementById()method of the Documentobject

•Sample HTML creating an inputelement with the idvalue firstName:<input type="text" id="firstName" />

•Sample JavaScript creating a variable that references that element:let fName= document.getElementById("firstName");

•Syntax for changing the value assigned to an attribute thus retrieved:document.getElementById("firstName").value = *value*;orfName.value= *value*;

Including a scriptelement for each code section

•Several scriptelements can be included in a single HTML file

•Statements in one script section are accessible to subsequent script sections

•E.g., variables declared in a script section can be used in subsequent script sections

•Placing the scriptelement

•Place script sections containing the document.write()method where the content is to be written

•The DOM is created during page load, and a script that references a part of the page that has not been loaded will cause an error

•Many developers place scripts at the end of the document to avoid this type of error

•The document.getElementById()method can reference page objects only after they are loaded into the DOM

Syntax for attaching a web page to a JavaScript source file from within the HTML file:<script src="*url*"></script>where urlis the JavaScript source file's name and location

•The scriptelement can be used for embedding JavaScript code *or*referencing a file—not both at once

Default behavior is for commands in an external .jsfile to be loaded when the browser initially encounters the scriptelement in the HTML file

•With the asyncattribute, the browser parses the HTML and JavaScript code together, only pausing to process the script

•With the deferattribute, the browser parses and loads the HTML, then processes the script

Placing JavaScript code in an external file is preferable because:

•Code can be shared among pages or among team members

•When shared among pages, code need only be downloaded once

•The HTML file is kept neater and cleaner

•Websites are easier to manage when the HTML, CSS, and JavaScript files each focus on one task

**Libraries**: JavaScript source files that store especially useful generic scripts used on many different websites

•You can incorporate a library into HTML code by creating a scriptelement in the head section and using the srcattribute to specify the file name

•Popular libraries include Node.js, jQuery, and Modernizr

•Developers usually create customized versions of large libraries containing only the code they need to limit download time

**Well formed** web page documents conform to the rules and requirements of HTML

•**Validating parser**: a program that checks whether a web page is well formed and whether the document conforms to a specific language definition known as DTD

•E.g., W3C Markup Validation Service at http://validator.w3.org/

•**Validation**: the process of verifying that your document is well formed and checking that the elements in your document are correctly written according to the element definitions in a specific DTD

•Embedding JavaScript in an XHTML document

•Enclose the scriptelement within a CDATA section

•**Character data (CDATA)**: a document section that is not interpreted as markup

•**Parsed character data (PCDATA)**: a document section that is interpreted as markup

Week2

**Function:** a programming structure consisting of a collection of statements that share a common purpose or calculate a value

Defining a function

•Syntax for a **named function**: function *functionName*(*parameters*) {*statements*}

•Syntax for an **anonymous function**: function (*parameters*) {*statements*}

Function’s **parameters** are the variables it uses

Enclosed in a **command block** (opening and closing curly braces)

JavaScript expression for **calling**a function: *functionName*(*paramValues*);

•paramValues passed to a function are the **arguments** (**actual parameters**)

Syntax for a function that returns a value: function *functionName*(*parameters*) {*statements*return *value*;}

•return statement ends execution and returns a single value

Using event handlers

Most direct method of associating a function with an event

•Drawback: places JavaScript code in the HTML file

•Syntax for creating an event handler as an attribute of the HTML element:

<*elem onevent*= "*function()*">

Events as object properties

•Places the event handler within the JavaScript code file

•Can only specify function name, not parameter values

•Only one function can handle an event at a time

•Syntax for an event as an object property: *object*.*onevent*= *function*;

Event listeners

•An **event listener** listens for an event as it propagates through a web page, during either:

•The **capture phase** (event moves down the object hierarchy) or

•The **bubbling phase** (event moves back up the object hierarchy)

•Can attach multiple functions to the same event

•Syntax for method that attaches an event listener to an object: *object*.addEventListener("*event*", *function*, *capture*)

Events and anonymous functions

•Include entire structure of anonymous function in place of function name in an event handler or event listener

•Can pass in parameter values with this approach

|  |  |
| --- | --- |
| **Function** | **Description** |
| decodeURI(*string*) | Decodes text strings encoded with encodeURI() |
| decodeURIComponent(*string*) | Decodes text strings encoded with encodeURIComponent() |
| encodeURI(*string*) | Encodes a text string so it becomes a valid URI |
| encodeURIComponent(*string*) | Encodes a text string so it becomes a valid URI component |
| eval (*string*) | Evaluates expressions contained within strings |
| isFinite(*number*) | Determines whether a number is finite |
| isNaN(*number*) | Determines whether a value is the special value NaN(Not a Number) |
| parseFloat(*string*) | Converts string literals to floating-point numbers |
| parseInt(*string*) | Converts string literals to integers |

Example of using built-in function to verify the socialSecurityNumber variable is not a number:

let socialSecurityNumber= "123-45-6789";

let checkVar= isNaN(socialSecurityNumber);

document.write(checkVar);

**Scope:** where a variable or function can be called within the program

•Variable/function is only recognized within scope

•Referencing elsewhere results in an error

Let and var declaration scopes

•Variables declared with let are **block scoped**: scope is limited to the command block

•Variables declared with var are **function scoped**: scope is limited to the function

Local and global scope

•Variables/functions with **local scope** (e.g., **local variables**) are accessible within the command block or function where they are defined

•Includes block scope and function scope

•Those with **global scope** (e.g., **global variables**) are defined outside a block/function and thus accessible throughout the program

Can create local and global variables with the same name but different values

•Local variable takes precedence when in scope

•Assigning a value to the local variable does not affect the global variable’s value outside the local variable’s scope

•Global variables most useful for small applications and variables used as constants

•Local variables preferable for values used within and changed by functions

let quantityPerBox= 12;

function describeCandy(productName, numberOfBoxes) {var numberCandies= quantityPerBox\* numberOfBoxes;

document.write("You would like " + numberCandies+ " of our " + productName+ " candies!");}quantityPerBox has global scope, unless this code appears within a command block in the larger program, in which case it would have block scope, a type of local scope. numberCandieshas function scope, a type of local scope. You can’t determine the scope of productNameor numberOfBoxesfrom this sample since they aren’t declared here.

**Data type:** the specific category of information that a variable contains

•**Primitive types:** data types that can be assigned only a single value

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| number | A positive or negative number with or without decimal places, or a number written using exponential notation |
| Boolean | A logical value of trueor false |
| string | Text such as “Hello World!” |
| undefined | An unassigned, undeclared, or nonexistent value |
| null | An empty value |

**Strongly typed** (**statically typed**) programming languages require that you declare the type of data that a variable contains and do not allow you to alter that type

•**Loosely typed** (**duck typed**, **dynamically typed**) programming languages do not require you to declare the data type and allow data types to be change

JavaScript is loosely typed

•Data types cannot be declared when variables are created

•JavaScript interpreter determines and assigns or reassigns the variable’s data type based on the type of data stored

Working with numeric values

•Integer: positive or negative number without decimal places

•**Floating point number:** positive or negative numbers containing decimal places

Working with numeric values

•Integer: positive or negative number without decimal places

•**Floating point number:** positive or negative numbers containing decimal places

Working with strings

•Text string: zero or more characters surrounded by double or single quotation marks

•**Empty string:** zero-length string value

•Can use quotation marks within strings:

document.write("Welcome to 'Fan Trick Photography’”);

document.write('Welcome to "Fan Trick Photography”’);

•To split a text string onto a new line without causing an error:

•Use two or more strings concatenated by the addition operator (+)

•For some browsers, end a line with the \character to indicate the string continues

•Create a **template literal** by enclosing the string in backtick characters (`)

Escape characters and sequences

•An **escape character** is placed before characters within strings to indicate that they are to be treated as regular keyboard characters, not as syntax

•JavaScript’s escape character is the backslash (\)

•**Escape sequence:** combination of an escape character with a specific character, usually to carry out a special function

|  |  |
| --- | --- |
| **Escape Sequence** | **Character** |
| \\ | Backslash |
| \b | Backspace |
| \r | Carriage return |
| \" | Double quotation mark |
| \f | Form feed |
| \t | Horizontal tab |
| \n | Newline |
| \0 | Null character |
| \' | Single quotation mark (apostrophe) |
| \v | Vertical tab |
| \x*XX* | Latin-1 character specified by the *XX*characters, which represent two hexadecimal digits |
| \u*XXXX* | Unicode character specified by the *XXXX*characters, which represent four hexadecimal |

Two types of JavaScript operators: binary and unary

•**Binary operator:** requires an operand before and after the operator

•**Arithmetic operators:** operators used to perform mathematical operations

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Returns** |
| + | Combines or adds two items | 12 + 3 | 15 |
| – | Subtracts one item from another | 12 –3 | 9 |
| \* | Multiplies two items | 12\*3 | 36 |
| / | Divides one item by another | 12/3 | 4 |
| % | Returns the remainder (**modulus**) after dividing one integer by another integer | 18%5 | 3 |
| \*\* | Raising a value to a power | 3\*\*2 | 9 |

**Unary operator:** requires just a single operand either before or after the operator

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Returns** |
| ++ | Increases a value by 1 | 12++ | 13 |
| –– | Decreases a value by 1 | 12–– | 11 |
| – | Changes the sign of a value | –12 | –12 |

Two types of unary operators:

•**Prefix operators**, which are placed before the variable

•**Postfix operators**, which are placed after the variable

•Prefix operator is applied before assignment operator:let x = 5;let y = ++x // x = 6 and y = 6

•Postfix operator is applied after assignment operator:

let x = 5;

let y = x++ // x = 6 and y = 5

An **assignment operator** (e.g., =) is used for assigning a value to a variable

•**Compound assignment operators** both assign a value and perform a calculation

•Interpreter will attempt to convert a nonnumeric to a numeric operand

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Equivalent to** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| –= | x –= y | x = x –y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x/y |
| %= | x %= y | x = x % y |
| \*\*= | x \*\*= y | x = x\*\*y |
|  |  |  |

**Comparison operators** (**relational operators**): used to compare two operands

•Two nonnumerical operands are compared in lexicographical order

•String plus number: interpreter converts string to number or returns false

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Description** |
| == | x == y | Tests whether x is equal in value to y |
| === | x === y | Tests whether x is equal in value to y and has the same data type |
| != | x != y | Tests whether x is not equal to y or has a different data type |
| !== | x !== y | Tests whether x is not equal to y and/or doesn’t have the same data type |
| > | x > y | Tests whether x is greater than y |
| >= | x >= y | Tests whether x is greater than or equal to y |
| < | x < y | Tests whether x is less than y |
| <= | x <= y | Tests whether x is less than or equal to y |

**Conditional operators** (**ternary operators**) return one of two possible values given the Boolean value of comparison

•Syntax: *condition*? *trueValue*: *falseValue*;

•Condition can be any expression that equals trueor false, including a Boolean variable

•Can return an expression instead of a value

Falsy and truthy values

•**Falsyvalues**, equivalent to false: ""(empty string), -0, 0, NaN, null, undefined

•Everything else is a **truthy value**, equivalent to true

•Can often use truthy and falsyvalues to make comparison operations more compact by omitting the comparison operator

**Logical operators**

•Used to combine expressions that will result in a Boolean value of trueor false

•Used for negating (swapping) a Boolean value

•Multiple conditions can be grouped within parentheses to create more complicated statements

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Definition** | **Example** | **Description** |
| && | and | (x === 5) && (y === 8) | Tests whether x is equal to 5 and y is equal to 8 |
| || | or | (x === 5) || (y === 8) | Test whether x is equal to 5 or y is equal to 8 |
| ! | not | ! (x < 5) | Test whether x is not less than 5 |

|  |  |  |
| --- | --- | --- |
| **Name** | **Special Operator** | **Description** |
| Property access | . | Appends an object, method, or property to another object |
| Array index | [] | Accesses an element of an array |
| Function call | () | Calls up functions or changes the order in which individual operations in an expression are evaluated |
| Comma | , | Separates multiple expressions in the same statement |
| Conditional expression | ?: | Executes one of two expressions based on the results of a conditional expression |
| Delete | delete | Deletes array elements, variables created without the varkeyword, and properties of custom objects |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Special Operator** | | **Description** | |
| Property exists | in | | Returns a value of trueif a specified property is contained within an object |
| Object type | instanceof | | Returns trueif an object is of a specified object type |
| New object | new | | Creates a new instance of a user-defined object type or a predefined JavaScript object type |
| Data type | typeof | | Determines the data type of a variable |
| Void | void | | Evaluates an expression without returning a result |

**Operator precedence** determines the order in which operations in an expression are evaluated

•**Associativity**determines precedence for operators with equal intrinsic precedence

•Examples:

•5 + 2 \* 8evaluates to 21

•30 / 5 \* 2evaluates to 12

•let x = 3;let y = 2;x = y \*= ++x; // Value of both x and y is 8

•(5 + 2) \* 8evaluates to 56

Accessing the **browser console** (**console**) displays error messages from the browser

•Locating an error in your program

•Browser console reports the line where detected each error is located

•Also reports lines that failed to run

•Be sure to make permanent corrections to code within your code editor