JAVASCRIPT CHAPTER 1

Syntax and variables

**Background**

In 1995, Brendan Eich created JavaScript so the Netscape Navigator browser could dynamically respond to user events. Ex: The web page's content could change when the user clicked a button or hovered over an image.

JavaScript was standardized by Ecma International in 1997 and called ***ECMAScript***. Ecma International continues to improve ECMAScript, releasing a new version each year. JavaScript is an implementation of the ECMAScript specification.

Today, JavaScript is one of the most popular programming languages. JavaScript is supported by every major web browser and makes web applications like Gmail and Google Maps possible. JavaScript is also popular outside the web browser. Ex: Node.js, which runs JavaScript, is a popular technology for creating server-side web applications.

JavaScript is executed by an interpreter. An ***interpreter*** executes programming statements without first compiling the statements into machine language. Modern JavaScript interpreters (also called ***JavaScript engines***) use ***just-in-time (JIT) compilation*** to compile the JavaScript code at execution time into another format that can be executed quickly.

ECMAScript name

*The name "ECMAScript" was a compromise between Netscape, Microsoft, and other organizations involved in the standardization of JavaScript. Brendan Eich once commented that "ECMAScript was always an unwanted trade name that sounds like a skin disease." Despite ECMAScript's similarity to eczema (a group of related skin diseases), the name has stuck.*

### Variables

A ***variable*** is a named container that stores a value in memory. A ***variable declaration*** is a statement that declares a new variable with the keyword ***let*** followed by the variable name. Ex: let score declares a variable named score.

A variable may be assigned a value after being declared. An ***assignment*** assigns a variable with a value, like score = 2. A variable may also be assigned a value on the same line when the variable is declared, which is called ***initializing*** the variable. Ex: let maxValue = 5; initializes maxValue to 5.

A variable may be assigned a value without first declaring the variable, but good practice is to always declare a variable before assigning a value to the variable.

A name created for an item like a variable is called an ***identifier***. JavaScript imposes the following rules for identifiers:

* An identifier can be any combination of letters, digits, underscores, or $.
* An identifier may not start with a digit.
* An identifier may not be a reserved word like let, function, or while.

A JavaScript coding convention is to name JavaScript variables with camel casing, where the identifier starts with a lowercase letter, and subsequent words begin with a capital letter. Ex: lastPrice is preferred over LastPrice or last\_price.

A ***constant*** is an initialized variable whose value cannot change. A JavaScript constant is declared with the ***const*** keyword. Ex: const slicesPerPizza = 8; creates a constant slicesPerPizza that is always 8.

var keyword

*A variable may also be declared with the var keyword, which is covered elsewhere in this material.*

**Data types**

Variables are not explicitly assigned a data type. JavaScript uses ***dynamic typing***, which determines a variable's type at run-time. Every variable has a data type, such as one of the data types in the table below.

Table 6.1.1: Example JavaScript data types.

| Data type | Description | Example |
| --- | --- | --- |
| ***string*** | Group of characters delimited with 'single' or "double" quotes | let name = "Naya";  let quote = 'He asked, "Shall we play a game?"'; |
| ***number*** | Numbers with or without decimal places | let highScore = 950;  let pi = 3.14; |
| ***boolean*** | true or false | let hungry = true;  let thirsty = false; |
| ***array*** | List of items | let teams = ["Broncos", "Cowboys", "49ers"]; |
| ***object*** | Collection of property and value pairs | let movie = { title:"Sing", rating:"PG" }; |
| ***undefined*** | Variable that has not been assigned a value | let message; |
| ***null*** | Intentionally absent of any object value | let book = null; |

**Comments and semicolons**

A ***comment*** is any text intended for humans that is ignored by the JavaScript interpreter. JavaScript uses the // and /\* \*/ operators to produce comments in code.

Figure 6.1.1: Comments.

*// Single line comment*

*/\* Multi-line*

*comment*

*\*/*

Feedback?

JavaScript does not require that statements be terminated with a semicolon. Only when two statements reside on the same line must a semicolon separate the two statements. Good practice is to avoid placing two statements on the same line. Some developers prefer to use semicolons at the end of statements, and others do not. Good practice is to consistently use semicolons or not throughout the code.

Figure 6.1.2: Using semicolons.

let totalPoints = 10;

*// No semicolon is required*

let totalLives = 3

*// Two statements on the same line require a semicolon*

totalPoints = 5; totalLives = 2

### Input and output

A JavaScript program may obtain text input from the user with the prompt() function. The ***prompt()*** function prompts the user with a dialog box that allows the user to type a single line of text and press OK or Cancel. The prompt() function returns the string the user typed or null if the user pressed Cancel.

Output may be produced using the function ***console.log()***, which displays text or numbers in the console. The ***console*** is a location where text output is displayed. Web browsers have a console (accessible from the brower's development tools) that displays output from code the browser executes. This chapter's activities display the console output in the web page.

**Arithmetic operators**

An ***expression*** is a combination of items like variables, numbers, operators, and parentheses, that evaluates to a value like 2 \* (x + 1). Expressions are commonly used on the right side of an assignment statement, as in y = 2 \* (x + 1).

An ***arithmetic operator*** is used in an expression to perform an arithmetic computation. Ex: The arithmetic operator for addition is +. JavaScript arithmetic operators are summarized in the table below.

Table 6.2.1: JavaScript arithmetic operators.

| Arithmetic operator | Description | Example |
| --- | --- | --- |
| + | Add | *// x = 3*  x = 1 + 2; |
| - | Subtract | *// x = 1*  x = 2 - 1; |
| \* | Multiply | *// x = 6*  x = 2 \* 3; |
| / | Divide | *// x = 0.5*  x = 1 / 2; |
| % | Modulus (remainder) | *// x = 0*  x = 4 % 2; |
| \*\* | Exponentiation | *// x = 2 \* 2 \* 2 = 8*  x = 2 \*\* 3; |
| ++ | Increment | *// Same as x = x + 1*  x++; |
| -- | Decrement | *// Same as x = x - 1*  x--; |

Expressions are computed using the same rules as basic arithmetic. Expressions in parentheses () have highest precedence, followed by exponentiation (\*\*). Multiplication (\*), division (/), and modulus (%) have precedence over addition (+) and subtraction (-). Ex: The expression 7 + 3 \* 2 = 7 + 6 = 13 because \* has precedence over +, but (7 + 3) \* 2 = 10 \* 2 = 20 because () has precedence over \*.

**Compound assignment operators**

A ***compound assignment operator*** combines an assignment statement with an arithmetic operation. Common JavaScript compound assignment operators are summarized in the table below.

Table 6.2.2: Compound assignment operators.

| Assignment operator | Description | Example |
| --- | --- | --- |
| += | Add to | *// Same as x = x + 2*  x += 2; |
| -= | Subtract from | *// Same as x = x - 2*  x -= 2; |
| \*= | Multiply by | *// Same as x = x \* 3*  x \*= 3; |
| /= | Divide by | *// Same as x = x / 3*  x /= 3; |
| %= | Mod by | *// Same as x = x % 4*  x %= 4; |

**Arithmetic with numbers and strings**

The + operator is also the string concatenation operator. ***String concatenation*** appends one string after the end of another string, forming a single string. Ex: "back" + "pack" is "backpack".

The JavaScript interpreter determines if + means "add" or "concatenate" based on the operands on either side of the operator. An ***operand*** is the value or values that an operator works on, like the number 2 or variable x.

* If both operands are numbers, + performs addition. Ex: 2 + 3 = 5.
* If both operands are strings, + performs string concatenation. Ex: "2" + "3" = "23".
* If one operand is a number and the other a string, + performs string concatenation. The number is converted into a string, and the two strings are concatenated into a single string. Ex: "2" + 3 = "2" + "3" = "23".

For all other arithmetic operators, combining a number and a string in an arithmetic expression converts the string operand to a number and then performs the arithmetic operation. Ex: "2" \* 3 = 2 \* 3 = 6.

The JavaScript functions ***parseInt()*** and ***parseFloat()*** convert strings into numbers. Ex: parseInt("5") + 2 = 5 + 2 = 7, and parseFloat("2.4") + 6 = 2.4 + 6 = 8.4.

If parseInt() or parseFloat() are given a non-number to parse, the functions return NaN. ***NaN*** is a JavaScript value that means Not a Number. Ex: parseInt("dog") is NaN. The JavaScript function ***isNaN()*** returns true if the argument is not a number, false otherwise. Ex: isNaN("dog") is true.