

# FLUENCY<sup>6</sup>

with information technology

SKILLS, CONCEPTS, & CAPABILITIES



LAWRENCE SNYDER

## Chapter 17

### *Fundamental Concepts Expressed in JavaScript*

# Table of Contents

- Part 1: Becoming Skilled at Computing
- Part 2: Algorithms and Digitizing Information
- Part 3: Data and Information
- Part 4: Problem Solving
  - Chapter 17: Fundamental Concepts Expressed in JavaScript
  - Chapter 18: A JavaScript Program
  - Chapter 19: Programming Functions
  - Chapter 20: Iteration Principles
  - Chapter 21: A Case Study in Algorithmic Problem Solving
  - Chapter 22: Limits to Computation
  - Chapter 23: A Fluency Summary

# Learning Objectives

- Tell the difference between [name, value, and variable](#)
- List 3 basic data types and the rules for specifying them in a program
- Explain the way in which the [assignment statement](#) changes a variable's value
- Learn expressions using [arithmetic, relational, and logical operators](#)
- Learn [conditional statements](#)
- Learn [compound statements](#)

# Programming Concepts [1/2]

- Programming is the **act of formulating an algorithm**
- Programming is a **systematic means** of solving a problem
  - A computer can follow the instructions and produce the intended result for **every input, every time**
  - **All steps** must be spelled out precisely and effectively
  - **All contingencies** must be planned for various situations
- Trying to program an algorithm precisely using English is **hopeless**
  - Natural languages are **too ambiguous** for directing anything as clueless as a computer

# Programming Concepts [2/2]

- Programming requires **thinking**
- Basic programming concepts provide **tools needed to formulate any computation based on thinking**
- This chapter 17 introduces the following programming concepts of **JavaScript**
  - Declarations of Variables
  - Data types, numbers, string literals, and Booleans
  - Expressions
  - Assignment
  - Conditionals (IF/Then)
- Chapter 18: **HTML+JavaScript**   Chapter19: **Functions**   Chapter20: **While (Loop)**

# Script Language

- 간단한 프로그래밍을 위해 **간단한 언어로 작성한 명령어 집합**을 의미한다.
  - 주로 보조적인 프로그래밍을 위해 사용한다
  - 일반적으로 응용 프로그램이나 유틸리티의 규칙(**rule**)과 구문(**syntax**)을 사용하여 표현된 **명령어들**과 **loop, IF/THEN**등 **단순한 제어 구조의** 조합으로 구성
- 응용 프로그램이나 운영 체제(**OS**)상에서 최종 사용자가 제어할 수 있는 절차를 조합한 일련의 처리를 자동화하기 위해 이용한다
- **웹환경이나 게임환경의 프로그래밍에서** 더욱 적극적으로 사용하는데, 이유는 개발 편의성을 위함이다.
  - **Web Designer, Game Designer, Game Artist**같은 **Non Programmer**들도 간단하게 배워서 쓸 수 있는 스크립트를 적극적으로 사용한다.
- **C#, PHP, JavaScript, Python, Ruby, Perl, Erlang, CoffeeScript.....**

# History of JavaScript

- 경쟁: [Java Consortium](#) vs [MicroSoft C++](#) (Visual Studio)
- Java Consortium needed [a lightweight interpreted language](#) in order to compete with [MicroSoft Visual Basic](#)
- [Brendan Eich \(Netscape\)](#) Livescript, Sept 1995 → [JavaScript](#), Dec 1995
- JavaScript is most commonly used as [part of web browsers](#), whose implementations allow [client-side scripts](#) to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed.
- JavaScript is classified as [a prototype-based scripting language](#) with dynamic typing and first-class functions. This mix of features makes it a [multi-paradigm language](#), supporting object-oriented, imperative, and functional programming styles.



### At the Espresso Stand

Espresso is concentrated liquid coffee. Some people enjoy drinking espresso straight, but others prefer a café latte, espresso in steamed milk; a cappuccino, espresso in equal parts of steamed milk and milk foam; or an Americano, espresso in near-boiling water. Espresso drinks are sold in three sizes: short (8 oz.), tall (12 oz.), and grande (16 oz.). These drinks are made with a standard amount of espresso, but coffee addicts often order additional shots of espresso. The price of the additional shots is added to the base price of the drink, and tax is figured in to produce the charge for the drink. The program to compute the price of an espresso drink is:

#### Input:

drink, a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

ounce, an integer, giving the size of the drink in ounces

shots, an integer, giving the number of shots

#### Output:

price in dollars of an order, including 8.8% sales tax

#### Program:

```
var price;
var taxRate = 0.088;
if (drink == "espresso")
    price = 1.40;
if (drink == "latte" || drink == "cappuccino") {
    if (ounce == 8)
        price = 1.95;
    if (ounce == 12)
        price = 2.35;
    if (ounce == 16)
        price = 2.75;
}
if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
price = price + (shots - 1) * .50;
price = price + price * taxRate;
```

**Figure 17.1** Sample JavaScript computation to figure the cost of espresso drinks.



# Variables

- Every entity in the real world situation has a name
- In programming terminology, the names are called variables whose values vary
- As the process evolves, the variables must refer to these new values
- The most commonly used programming language operation is the command to change the value of a variable (called Assignment statement)
  - Ex. Current-US-President = “Bill Clinton” // at 2008  
Current-US-President = “Barack Obama” // at 2015

Name	Current Value (7/1/2011)	Previous Values
U.S. President	Barack Obama	Bill Clinton, George H. W. Bush
Chief Justice U.S. Supreme Court	John Roberts	Warren Burger, Earl Warren
James Bond	Daniel Craig	Sean Connery, Roger Moore
Queen of England	Elizabeth II	Victoria I, Elizabeth I
U.N. Secretary General	Ban Ki-moon	Boutros Boutros-Ghali, Kofi Annan

# Identifiers and Their Rules

- Identifier is the letter sequence that makes up a variable
  - must begin with a letter, followed by any sequence of letters, numerals, or the underscore symbol
  - are not allowed to contain space
  - The underscore symbol can be used as a word separator
  - Identifiers are case sensitive (i.e. uppercase and lowercase letters are different)
  - Example: current\_freshman, freshmen\_2015, Freshmen\_2015

# Variable Declaration Statement

- The first thing to do in programming is to declare the variables that will be used later
- Declaring variables is done using a command called a declaration
- In JavaScript, the declaration command is the word `var`, followed by a list of the variables to be declared, separated by commas
  - `var area, radius;`
- The statement terminator in JavaScript is the semicolon “;”
- JavaScript allows declaration statements anywhere in the list of statements

# Initializing a Declaration

- **Setting the initial value** as part of the declaration (called **initializing the variable**)

```
var taxRate = .088;
```

```
var balanceDue = 0;
```

```
var temperature = 0;
```

```
var humidity_rate = 0;
```

- Logically related variables are declared **in a single declaration statement**

```
var taxRate = .088, balanceDue = 0;
```

```
var temperature = 0, humidity_rate = 0;
```

- Variable may not have an initial value

```
var Currentscore;    # Currentscore has an undefined value
```

# 3 Basic Data Types of JavaScript

- The data type mean the different kinds of values of a programming language
- There are 3 basic data types used here for JavaScript
  - Numbers → 2, 5, 17.8, etc
  - Strings → 'HJ Kim', "SNU", etc
  - Booleans → true, false
- Of course, there are several other data types

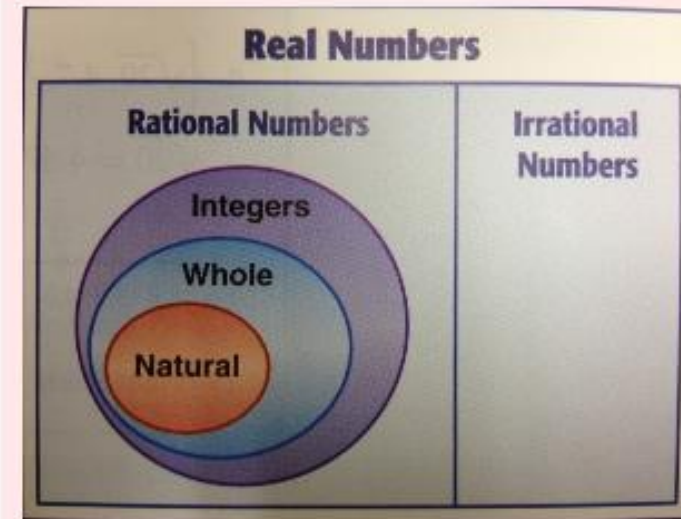
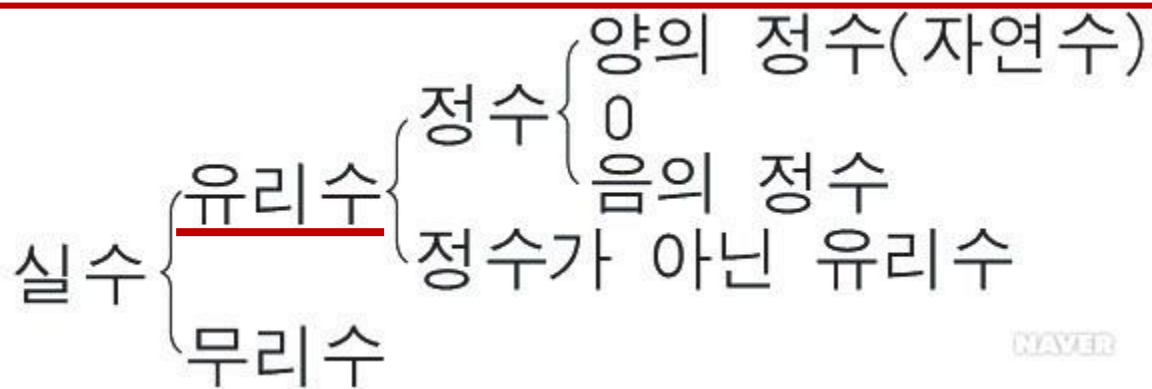
# Rules for Writing Numbers in JavaScript

- One “unusual” aspect of numbers in programming is that there are **no “units”**
  - Numbers must be written in decimal form (0.33, not 33%; 10.89, not \$10.89)
- Standard computer numbers
  - Have about 10 significant digits
  - Range from as small as  $10^{-324}$  to as large as  $10^{308}$
  - Numbers and computer arithmetic are **unexpectedly subtle**
- As a general rule, the “**safe zone**” for numbers is the range from 2 billionths to 2 billion plus or minus →  $1/(2,000,000,000) \sim 2,000,000,000$

만약 1bit를 부호로 쓰고, 31bits를  
2진숫자의 저장으로 쓴다면

**In fact, it is not that simple!**

# Number System



실수 = Real Number    유리수 = Rational Number

**A rational number** is any [number](#) that can be expressed as the [quotient](#) or fraction  $p/q$  of two [integers](#),  $p$  and  $q$ , with the [denominator](#)  $q$  not equal to zero

Decimal = 10진법 숫자

decimal fractions  $8/10$ ,  $1489/100$ ,  $24/100000$ , and  $58900/10000$  are expressed in decimal notation as 0.8, 14.89, 0.00024, 5.8900 respectively.

A irrational number needs to be approximated to a rational number

# 컴퓨터의 숫자표현: Only Integers

32 bits



64 bits



## Ranges for typical computer “word” sizes

<u>bits</u>	<u>minimum</u>	<u>maximum</u>
8	0	$2^8 - 1$ (255)
16	0	$2^{16} - 1$ (65,535)
32	0	$2^{32} - 1$ (4,294,967,295)
64	0	$2^{64} - 1$ (18,446,744,073,709,551,615)

<u>bits</u>	<u>minimum value</u>	<u>maximum value</u>
8	$-2^7 = -128$ 10000000	$2^7 - 1 = +127$ 01111111
16	$-2^{15} = -32,768$	$2^{15} - 1 = +32,767$
32	$-2^{31}$ $= -2,147,483,648$	$2^{31} - 1$ $= +2,147,483,647$
64	$-2^{63}$ $= -9,223,372,036,854,775,808$	$2^{63} - 1$ $= +9,223,372,036,854,775,807$



## 컴퓨터의 Rational Number 표현: Fixed point representation

 $\pm z z z z . f f f f$ Integer 표현틀에서  $z, f$  는 0 또는 1

64 bits



## How to Compute Fractional Number

*Q m.n Format*

$b'_s b'_{m-1} \dots b'_0$	$b_{n-1} b_{n-2} \dots b_0$
----------------------------	-----------------------------

$$-2^m b'_s + \dots + 2^1 b'_1 + 2^0 b'_0 + 2^{-1} b_{n-1} + 2^{-2} b_{n-2} \dots + 2^{-n} b_0$$

Examples:

- 1110 Integer Representation Q3.0:  $-2^3 + 2^2 + 2^1 = -2$
- 11.10 Fractional Q1.2 Representation:  $-2^1 + 2^0 + 2^{-1} = -2 + 1 + 0.5 = -0.5$   
(Scaling by  $1/2^2$ )
- 1.110 Fractional Q3 Representation:  $-2^0 + 2^{-1} + 2^{-2} = -1 + 0.5 + 0.25 = -0.25$  (Scaling by  $1/2^3$ )

## 컴퓨터의 Rational Number 표현: Floating point representation

Floating point is based on scientific notation

Age of the Universe in years:

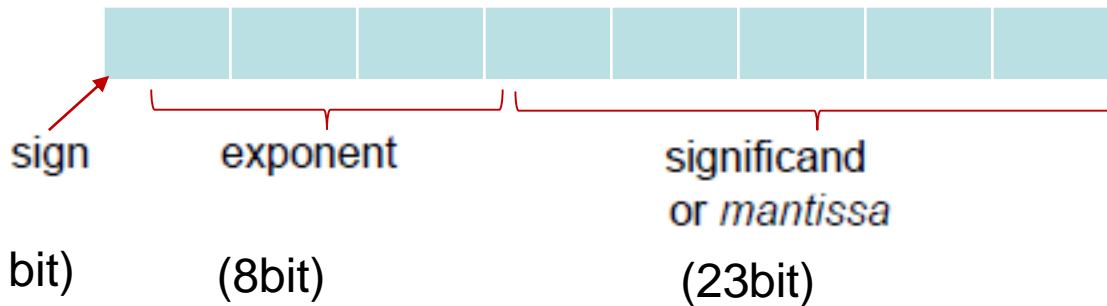
$+ 1.37 \times 10^{10}$

sign

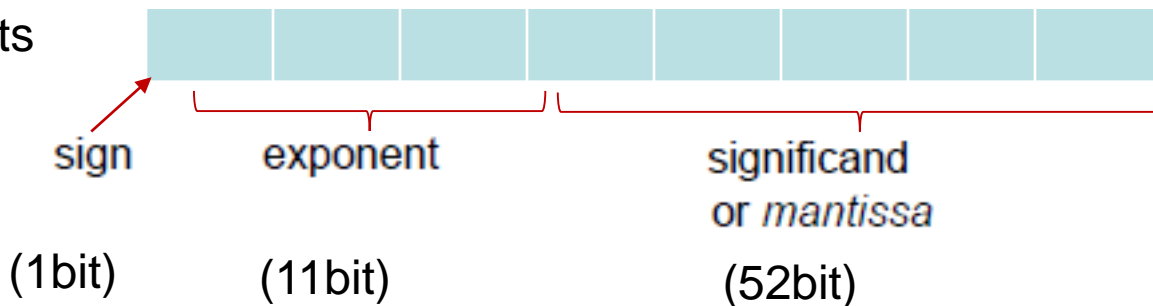
significand  
or *mantissa*

exponent

32 bits



64 bits



232는

$23.2 \times 10$

$2.32 \times 100$

$0.232 \times 1000$

으로 볼수 있는데


$2.32 \times 100$ 으로 정규화

→ 모든수의 소수점을  
이동시켜서 정규화하므로  
floating point representation

# Strings in JavaScript

- Strings are “sequences of keyboard characters” for representing text
  - Notice that a string is always surrounded by single (') or double (") quotes
- To use double quotes in a string, enclose the string in single quotes:  
`var answer = 'He said, "No!" '`
- If our string contains single quotes, enclose it in double quotes:
- `var answer = " He said, 'STOP' "`
- Since the apostrophe is commonly used in possessives and contractions, use double quotes as the default  
`var book = "Guide to B&B's"`

# Rules for Writing Strings in JavaScript

- Strings must be surrounded by **quotes**, either single (') or double ("), **which are not curly** ( “curly” **vs** "not\_curly")
  - NOTEPAD2 will give you only **not-curly double quotes**
- Most characters are allowed within quotes **except**:
  - the return character () , **backspace character**, **tab character**, **\**, and **two others**
- Any number of characters is allowed in a string
- The minimum number of characters in a string is zero (""), which is called **the empty string**  
  
`var starting_pitcher = "";`
- The characters that are typed in the program are also called **literals**
  - So, numbers and strings are literals

# String constants or String literals

- The quotes (ex. "snu") are removed when the string literal is stored in the computer
- Any character can be stored in the computer's memory
- Prohibited characters can be the value of a string in the computer by using the "escape" mechanism

## Escape Mechanisms

- For JavaScript, the escape symbol is the backslash (\)
- The escape sequences are converted to the single characters they represent when stored in the computer's memory

```
var fourTabs = "\t\t\t\t", backUp = "\b",  
    bothQuotesInOne = "\" \"";
```

Table 17.1 Escape sequences for characters prohibited from string literals.

Sequence	Character	Sequence	Character
\b	Backspace	\f	Form feed
\n	New line	\r	Carriage return
\t	Tab	\'	Apostrophe or single quote
\"	Double quote	\\	Backslash

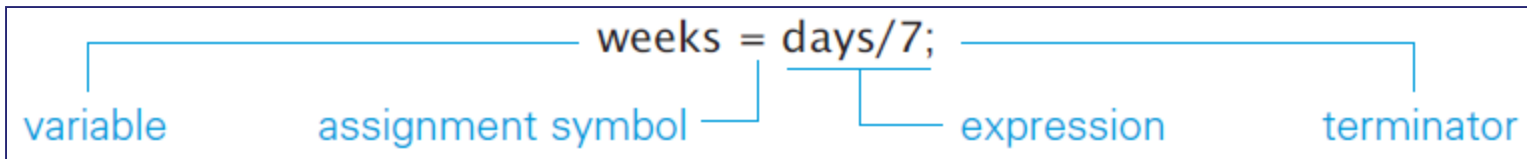
# Boolean Values

- Boolean values sometimes called **Booleans**
  - only 2 Boolean values: **true** and **false**
  - they are “**reserved**” **values**, not identifiers or strings
- So, don't be confused when you see something like **mers\_test = false**

```
var foreignLanguageReq = false, mathReq = true, totalCredits = 0;
```

# The Assignment Statement

- The assignment statement **changes a variable's value**
- An assignment statement has 3 parts that always occur in this order:  
**<variable> <assignment symbol> <expression>;**
- The assignment statement is terminated by a semicolon
- JavaScript's <assignment symbol> is **the equal sign (=)**



- Different PLs use different symbols for indicating assignment:
  - The equal sign (=)
  - The colon/equal sign pair (:=)
  - The left pointing arrow ( $\leftarrow$ )

# Interpreting an Assignment Statement

- A value flows from the right side (expression side) to the left side (variable side)
- The assignment symbol should be read as “is assigned ” or “becomes” or “gets”
- In an assignment statement everything to the right of the assignment symbol is computed/evaluated first
- If there are any variables used, their current values are used
- Ex. `weeks = days/7;`
  - The current value of `days` is retrieved from memory
  - That value is divided by `7`
  - The answer becomes the new value of the variable `weeks`



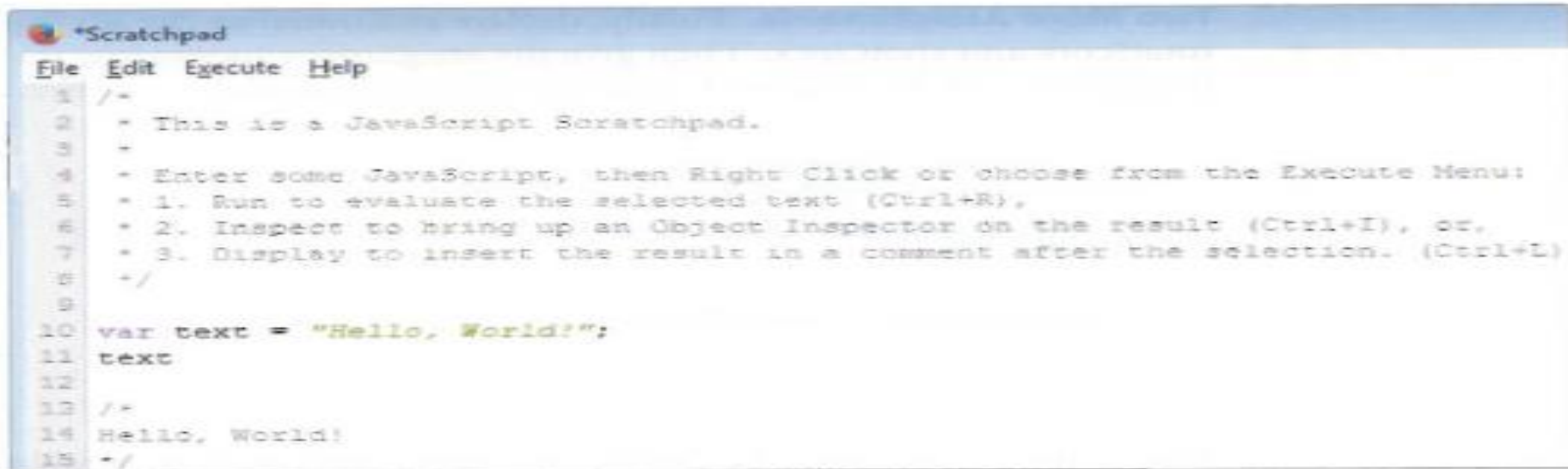
# Lab Practice [1/2]

- We can try everything we learn in [the Firefox browsers](#)
- [Scratchpad](#) in FireFox allows us to type in JavaScript and run it
  - FireFox > Developer > Scratchpad
    - Run to evaluate the selected text (Ctrl+R)
    - Display to insert the result in a comment after the selection. (Ctrl+L)
  - After finishing JavaScript coding, we will embed the [JavaScript code](#) inside [HTML](#)!
- Comments in JavaScript
  - Multiline comments begin with `/*` and end with `*/`
  - One line comment by using `//`
- Exercise: Change days to be initialized to some other number, instead of 77 and display the new result

```
10 var days = 77;           //Declare and initialize
11 var weeks;               //Declare
12
13 weeks = days/7;
14
15 weeks
16
17 /*
18 11
19 */
```

# JavaScript Editor inside Browser

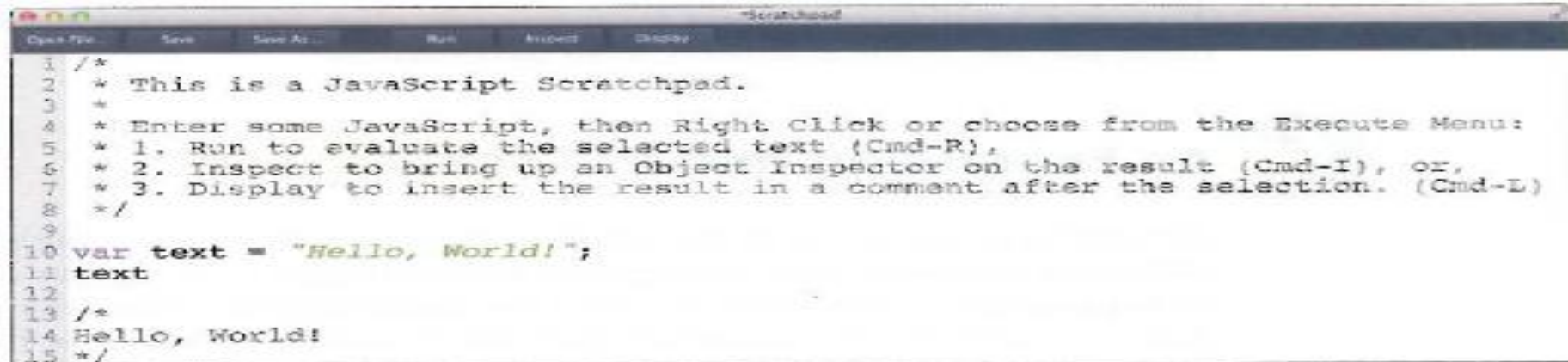
- Scratchpad (Firefox) and JavaScript Console (Chrome)



The screenshot shows the Firefox Scratchpad window for Windows. The title bar is blue and says '\*Scratchpad'. The menu bar includes 'File', 'Edit', 'Execute', and 'Help'. The code editor contains the following text:

```
1 /*
2  * This is a JavaScript Scratchpad.
3  *
4  * Enter some JavaScript, then Right Click or choose from the Execute Menu:
5  * 1. Run to evaluate the selected text (Ctrl+R),
6  * 2. Inspect to bring up an Object Inspector on the result (Ctrl+I), or,
7  * 3. Display to insert the result in a comment after the selection. (Ctrl+L)
8  */
9
10 var text = "Hello, World!";
11 text
12
13 /*
14 Hello, World!
15 */
```

(a)



The screenshot shows the Firefox Scratchpad window for MacOS. The title bar is grey and says '\*Scratchpad'. The menu bar includes 'Open File...', 'Save', 'Save As...', 'Run', 'Inspect', and 'Display'. The code editor contains the following text:

```
1 /*
2  * This is a JavaScript Scratchpad.
3  *
4  * Enter some JavaScript, then Right Click or choose from the Execute Menu:
5  * 1. Run to evaluate the selected text (Cmd-R),
6  * 2. Inspect to bring up an Object Inspector on the result (Cmd-I), or,
7  * 3. Display to insert the result in a comment after the selection. (Cmd-L)
8  */
9
10 var text = "Hello, World!";
11 text
12
13 /*
14 Hello, World!
15 */
```

(b)

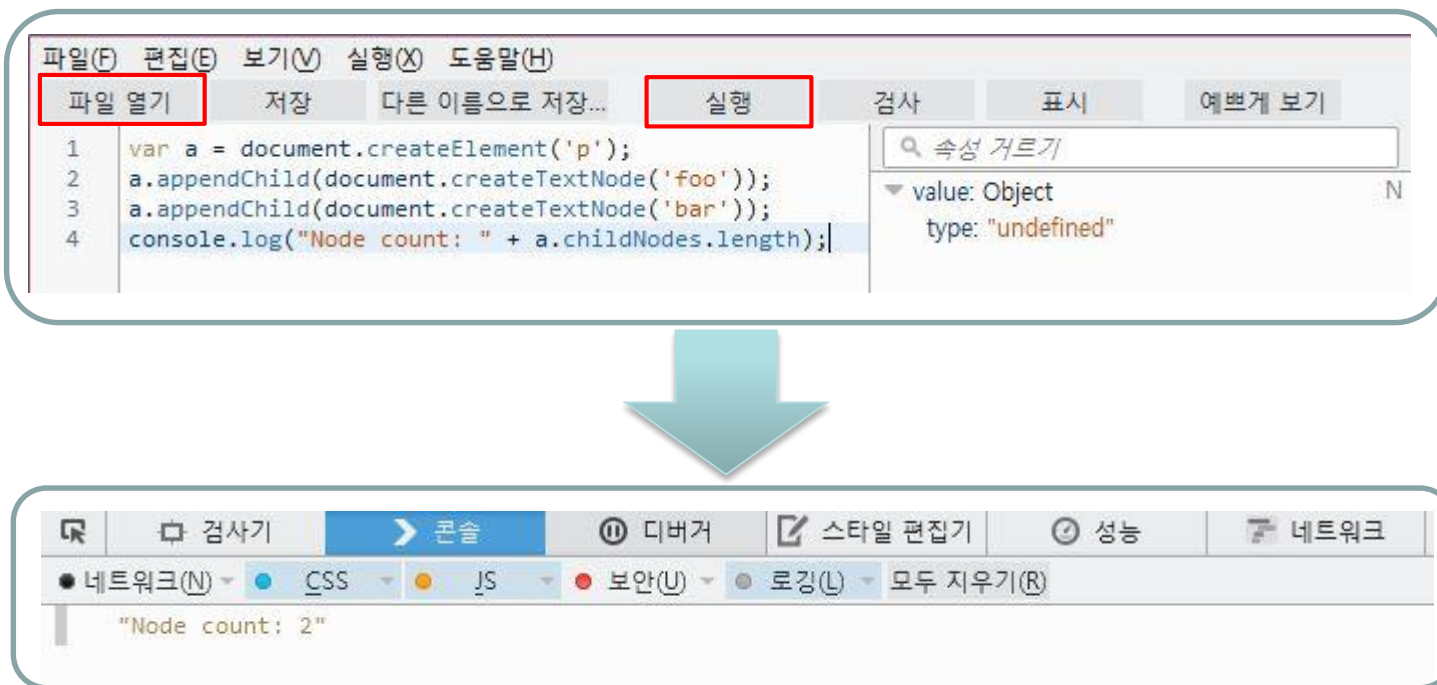
**Figure 17.2** Firefox's Scratchpad development page: (a) for Windows OS, and (b) for MacOS.

# Editing with Complex JavaScript Functions

- These features uses Console API
  - Measuring how long something takes → `console.time()`, `console.timeEnd()`
  - Writing to the console log → `console.log()`
  - Errors and Warnings → `console.error()`
  - Assertions → `console.assert()`
  - Grouping Output → `console.group()`, `console.groupEnd()`
  - Viewing Structured Data → `console.table()`
- These features uses Command line API
  - Controlling the CPU Profiler → `profile()`, `profileEnd()`
  - Monitoring Events → `monitorEvents()`

# Scratchpad (FireFox)

- Scratchpad can execute multiple lines of JS code
  - Can import .js files into the browser and execute it



## Lab Practice [2/2]

```
1 var totalScore = 10;
2 var shotClock = 10;
3
4 totalScore = totalScore + 3; //now typing ^L
5 /*
6 13
7 */
8 shotClock = shotClock - 1; //now typing ^L
9 /*
10 9
11 */
```

- Initialize 2 more variables (totalScore, and shotClock) to 10
- Then write 2 assignment statements, one that adds 3 to totalScore and the other that subtracts 1 from shotClock
- To Save your work type ^S, then save the file with .js extension, as in `basketball.js`

# An Expression and Its Syntax

- Programming is not mathematics but it has its roots in mathematics
- One programming concepts is an algebra-like formula called an expression
  - Expressions describe the means of performing an actual computation
  - Expressions are built of variables and operators
- Expressions usually follow rules similar to algebraic formulas
- Multiplication must be given explicitly with the asterisk (\*):  $a * b$
- \* and / are performed before (have a higher precedence than) + and -
  - Parentheses can bypass that order

# Some of JavaScript Arithmetic Operators

- Superscripts ( $x^2$ ) are prohibited in JavaScript, instead use  $x * x$ 
  - Some languages have an operator for exponents or powers, but not in JavaScript
- $*$ ,  $/$ ,  $+$  and  $-$  are called **binary operators** operated on **two operands**
- **Unary operators** has only one operand (Ex. Negate  $-$ )
- Another useful operator is **mod** (%)
- The result of  $a \% b$  for integers  $a$  and  $b$  is the remainder of the division  $a/b$ 
  - Examples:
    - $4 \% 2$  is 0 because 2 evenly divides 4
    - $5 \% 2$  is 1 because 2 into 5 leaves a remainder of 1

# Relational Operators

- Sometimes, Comparison operators
- The relationship between two numbers is tested by comparisons in relational operators (normally 6 types)
- The outcome of the comparison is a Boolean value of true or false
- The “equal to” relational operator (==) is a double equal sign
- The “not equal to” operator uses the !=

$a < b$	Is a less than b?
$a \leq b$	Is a less than or equal to b?
$a == b$	Is a equal to b?
$a != b$	Is a not equal to b?
$a \geq b$	Is a greater than or equal to b?
$a > b$	Is a greater than b?



# Logical Operators [1/2]

- The relational test results in a true or false outcome
- It is common to test 2 or more relationships together
  - This requires that relational expression results should be combined
  - Logical operators have lower precedence than the relational operators
- Logical AND Operator: &&
  - It plays the same role as AND plays in query expressions
  - The outcome of a && b is true if both a and b are true; otherwise, it is false
  - The operands a and b can be variables, or expressions, or a mix

Value of Age	Age > 12	Age < 20	Age > 12 && age < 20
4	false	true	false
16	true	true	true
50	true	false	false

# Logical Operators [2/2]

- Logical OR Operator: ||

- a || b is true if either a is true or b is true
- a || b is false only if both are false

`(age == 10 || age == 11) || age == 12`

- Logical NOT operator: !

- It is a unary operator, taking only a single operand
- Its outcome is the opposite of the value of its operand
- By placing the logical not operator in front of the parenthesized expression, we have a new expression with the opposite outcome

`!(age > 12 && age < 20)`

# Operator Overloading

- Operator overloading is a technical term meaning the “use of an operator with different data types”
- Operators usually apply to a single data type
  - $4 + 5$  produces the numerical result of 9
- If operands are strings, what does the  $+$  mean? ➔ String concatenation
  - “Seoul” + “University” produces “SeoulUniversity”
- The meaning of  $+$  is overloaded:
  - $+$  to mean addition when operands are numeric
  - $+$  means concatenation when the operands are strings

# Conditional Statements

- A conditional statement or a conditional makes testing numbers and strings **simple**
- The conditional has the form:  
**if (<Boolean expression>)**  
    **<then-statement>;**
  - The <Boolean expression> is an expression evaluating to true or false
  - The <then-statement> is any JavaScript statement and normally indented
- The <Boolean expression> is called **a predicate**
  - If the outcome is true, the <then-statement> is **performed**
  - If the outcome is false, the <then-statement> is **skipped**
  - Ex: **if (waterTemp < 32)**  
    **waterState = "Frozen";**

# Compound Statements

- Programming languages allow for a sequence of statements in the <then-statement>
- Group multiple statements by surrounding them with “curly braces” { }
- { } collects single statements together to become a compound statement
  - The opening { is placed immediately after the predicate to signal that a compound statement is next
  - The closing } is placed conspicuously (눈에 잘 보이게) on its own line after the last statement within the compound
  - The closing } should not be followed by a semicolon

```
if (waterTempC < 0) {  
    waterState = "Frozen";  
    description = "Ice";  
}
```

```
if (waterTempC < 0) {  
    waterState = "Frozen";  
}
```

This is OK, too!

# if/else Statements

- The if/else statement contain statements that will be executed when the condition's outcome is false

if (<Boolean expression>)

    <then-statement>;

else

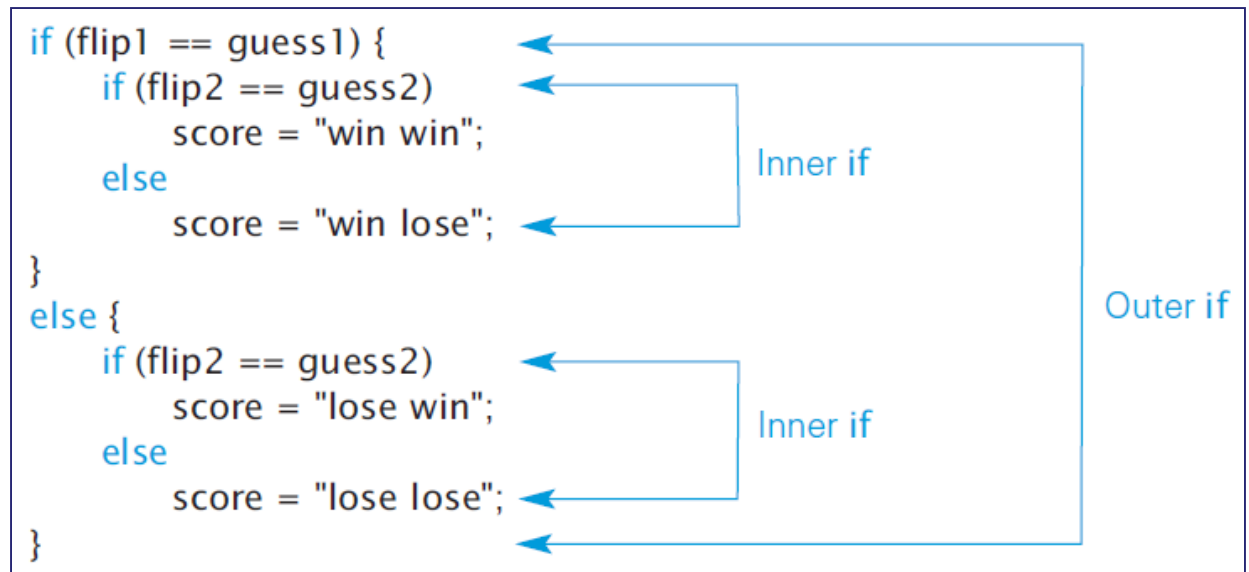
    <else-statement>;

```
1 var day = "Tuesday";
2 var calendarEntry;
3
4 if (day == 'Friday' || day == 'Saturday')
5     calendarEntry = "Party!";
6
7 else
8     calendarEntry = "Study";
9
10 calendarEntry
11 /*
12 Study
13 */
14 */
```

- The <Boolean expression> is evaluated first
  - If the <Boolean expression>'s outcome is **true**:
    - The <then-statement> is executed and the <else-statement> is skipped
  - If the <Boolean expression>'s outcome is **false**:
    - The <then-statement> is skipped and the <else-statement> is executed

# Nested if/else Statements

- Both the <then-statement> and the <else-statement> can contain an if/else
- The rule in JavaScript and most other PLs is that **the else** associates with **the (immediately) preceding if**
- This can be confusing to read
- The best policy is to enclose the <then-statement> or <else-statement> in **compound curly braces** whenever they contain an if/else



# The Espresso Program [1/7]

- The program computes the price of four kinds of espresso drinks based on:

- The type of drink
- The size of drink
- The number of additional shots
- Plus tax

## Input:

drink—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

ounce—an integer, giving the size of the drink in ounces

shots—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```



# The Espresso Program [2/7]

- The input variables are listed at the start of the program, as is the output
- The input variables are assumed to be given
- The program will create the output

## Input:

drink—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"  
ounce—an integer, giving the size of the drink in ounces  
shots—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# The Espresso Program [3/7]

- The output is declared to be a **variable** in the first statement of the program.
- Statements 3 through 5 determine the kind of drink and establish the base price

## Input:

drink—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

ounce—an integer, giving the size of the drink in ounces

shots—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# The Espresso Program [4/7]

Suppose we purchase a **double tall latte**

- Line 3 is executed:

The test `drink == "espresso"`

fails, because the variable

`drink` has the value `"latte"`

The `then` statement is skipped

## Input:

`drink`—a character string with one of the values: `"espresso"`, `"latte"`, `"cappuccino"`, `"Americano"`

`ounce`—an integer, giving the size of the drink in ounces

`shots`—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# The Espresso Program [5/7]

Suppose we purchase a **double tall latte**

- Line 4 is executed:

The test `drink == "latte" || drink == "cappuccino"` has a true outcome

- Line 4a is executed:

The test `ounce == 8` has a false outcome, so its then statement is skipped

- Line 4b is executed:

The `ounce == 12` test is true, so the then statement is executed

- Line 4c is executed:

The `ounce == 16` test fails, so its then statement is skipped

## Input:

`drink`—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

`ounce`—an integer, giving the size of the drink in ounces

`shots`—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# The Espresso Program [6/7]

Suppose we purchase a **double tall latte**

- Line 5 is executed:

The `drink == "Americano"` test fails, so its then statement is skipped

- Line 6 is executed:

This causes the value of shots minus 1 to be multiplied by .50, resulting in the value .50, which is added to price, yielding price  $\Leftarrow \Rightarrow$  2.85

## Input:

drink—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

ounce—an integer, giving the size of the drink in ounces

shots—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# The Espresso Program [7/7]

Suppose we purchase a **double tall latte**

- **Line 7 is executed:** Current value of price is multiplied by taxRate, resulting in 0.25, which is added to price to compute the final value of 3.10, which is assigned to price

## Input:

drink—a character string with one of the values: "espresso", "latte", "cappuccino", "Americano"

ounce—an integer, giving the size of the drink in ounces

shots—an integer, giving the number of shots

## Output:

price in dollars of an order, including 8.8% sales tax

## Program:

```
1.  var price;
2.  var taxRate = 0.088;
3.  if (drink == "espresso")
    price = 1.40;
4.  if (drink == "latte" || drink == "cappuccino") {
4a.    if (ounce == 8)
        price = 1.95;
4b.    if (ounce == 12)
        price = 2.35;
4c.    if (ounce == 16)
        price = 2.75;
    }
5.  if (drink == "Americano")
    price = 1.20 + .30 * (ounce/8);
6.  price = price + (shots - 1) * .50;
7.  price = price + price * taxRate;
```

# Summary [1/2]

- Variables can be **initialized** when they are declared
- Changing the value of a variable is possible by using **assignment**
- **An assignment statement** has a variable on the left side of the symbol and an expression on the right side
  - This makes information flow from **right to left in an assignment statement**
  - Statements like  **$x = x + 1$** ; make sense in programming, but not in algebra
- There are **3 JavaScript data types**: numbers, strings, and Booleans
- Standard arithmetic operators and relationals compute on numbers, and logical operations on Booleans

# Summary [2/2]

- As a rule, all programming statements are executed one after another, starting at the beginning
  - JavaScript's 2 conditional forms are `if` and `if/else`
  - These allow statements to be executed depending on `the outcome of a Boolean expression` called `a predicate`
- `The espresso program` illustrates most of the ideas in this chapter
- All that keeps us from running the program and demonstrating our knowledge is setting up the input to acquire the values for drink, ounce, and shots, and outputting the price. This requires `a UI written in HTML` (the topic of Chapter 18)