Introduction To Full-Stack Web Development

CS 386

Michael Kremer

Last updated: 9/24/2023 9:04:34 AM

- 9.1 Core JavaScript
- 9.2 Data Types
- 9.3 Values and Conversion

Class 9

9.1 Core JavaScript

- JavaScript is hosted programming language
- Meaning it does not exist in its own and must be executed in host environment
- Most common host environment for JavaScript is web browser
- Core JavaScript language defines minimal API (Application Programming Interface):
    - For working with text, arrays, dates, and regular expressions
    - But does not include any input or output functionality

9.1 Core JavaScript

- Host environment is responsible:
  - Input and output
  - More sophisticated features such as networking, storage, and graphics
- This is client-side JavaScript
- To learn, write, and execute JavaScript, need environment to test JavaScript
- Most browsers have built-in tools to execute JavaScript
- Countless numbers of authoring tools for JavaScript:
  - Web browser tools
  - Simple text editors, either OS built-in or more sophisticated ones Notepad++)
  - Integrated Development Environment (IDE) tools (Visual Studio, Eclipse, etc.)

9.1 Core JavaScript

- Core JavaScript language includes following basic components:
  - ❑ Lexical structure
  - ❑ Types, Values, Variables
  - ❑ Expressions and Operators
  - ❑ Statements
  - ❑ Objects
  - ❑ Arrays
  - ❑ Functions
  - ❑ Pattern Matching with Regular Expressions
- Client-side JavaScript technology in modern web browsers:
  - ❑ JavaScript in Web Browsers
  - ❑ Window object
  - ❑ DOM object
  - ❑ Scripting CSS
  - ❑ Handling Events
  - ❑ jQuery library

9.2 Data Types

- JavaScript types can be divided into two main categories:
  - primitive types: numbers, strings, Booleans, JavaScript special values (null, undefined)
  - object types (unordered collection of named values, arrays, function, dates)
- Any JavaScript value that is not number, string, Boolean, or null or undefined is of object type
- Object (that is, member of type object):
  - Collection of properties
  - Each property has name and value (either primitive value, such as number or string, or object)

9.2 Data Types

- Ordinary JavaScript object is unordered collection of named values
- JavaScript built-in objects:
  - Arrays
  - Dates
  - Functions
- Array is special kind of object → ordered collection of numbered values
- JavaScript defines another special kind of object, known as function
- Function is object that has executable code associated with it
- Function may be invoked to run that executable code and return computed value
- Like arrays, functions behave differently from other kinds of objects
  - JavaScript defines special language syntax for working with them

9.2 Data Types

- Numbers
  - Unlike many languages, JavaScript does not make distinction between integer values and floating-point values
  - All numbers in JavaScript are represented as floating-point values
  - JavaScript represents numbers using 64-bit floating-point format defined by IEEE 754 standard:
    - Can represent numbers in range of:
      - $\pm 5 \times 10^{-324}$ to $\pm 1.7976931348623157 \times 10^{308}$ (smallest to largest)
  - Number format allows to exactly represent all integers between:
    - −9007199254740992 (-2^{53}) and 9007199254740992 (2^{53})
  - For floating point decimals, simply use period as decimal point
  - For exponential notation, use e or E:
    - 6.02e23 or 6.02E23

9.2 Data Types

- Numbers
- Arithmetic operators:
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % modulo (remainder after division)
- Math library includes many mathematical functions and constants
- Examples:
  - Math.sqrt(2) = 1.4142135623730951
  - Math.pow(2,3) = 8 (2^3)

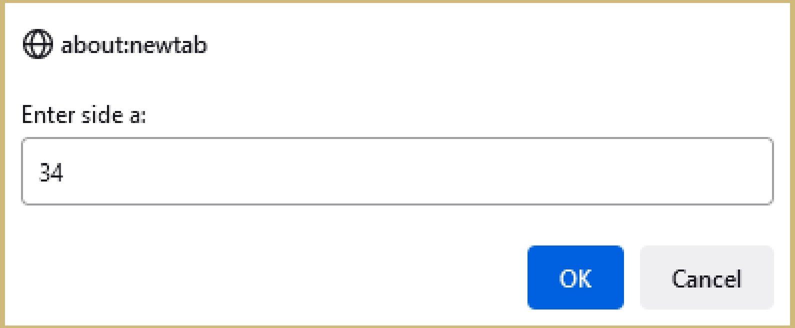
Syntax:
Math.*function*

9.2 Data Types

- Numbers
  - Arithmetic in JavaScript does not raise errors in cases of:
    - Overflow
    - Underflow
    - Division by zero
  - When result of numeric operation is larger than largest representable number (overflow):
    - Result is special infinity value, which JavaScript prints as Infinity
    - Also negative infinity
  - Division zero by zero not well defined:
    - Returns NaN (not a number)
  - Also square root of negative number returns NaN

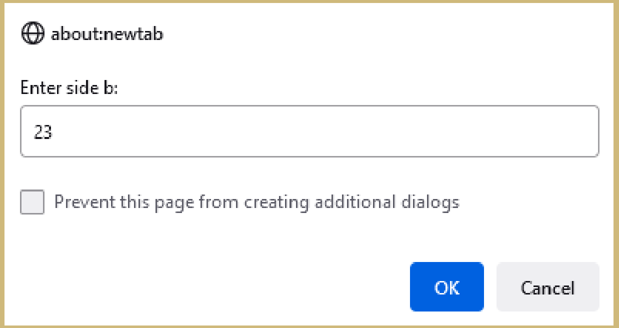
9.2 Data Types

- **Example 9 -1:**
  - Calculate hypotenuse of right-angled triangle
  - Solicit two sides of triangle from user using prompt (default 0) and store in variables a and b
  - Calculate hypotenuse using math library functions sqrt and pow, store in variable hypo
  - Display result in alert and console, execute in browser


about:newtab

Enter side a:

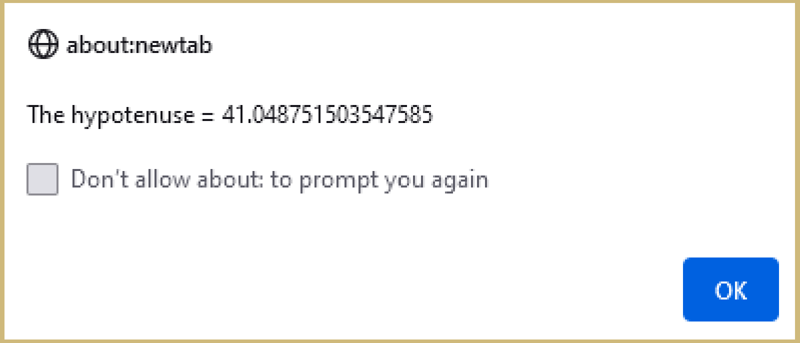
34


about:newtab

Enter side b:

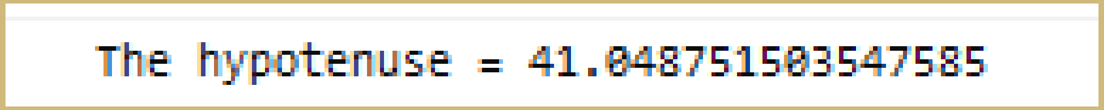
23

☐ Prevent this page from creating additional dialogs


about:newtab

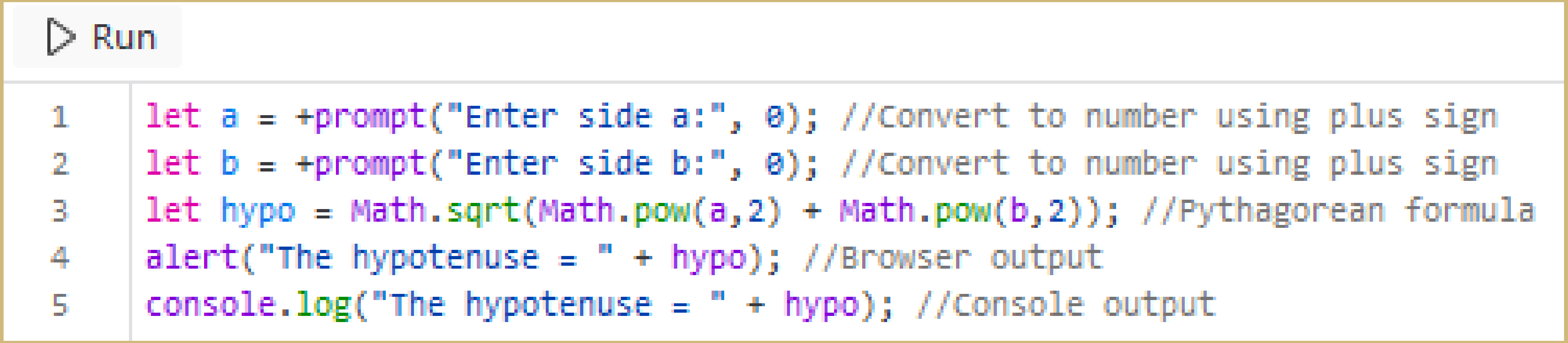
The hypotenuse = 41.048751503547585

☐ Don't allow about: to prompt you again


```
The hypotenuse = 41.048751503547585
```

9.2 Data Types

- **Example 9-1:**



```
let a = +prompt("Enter side a:", 0); //Convert to number using plus sign
let b = +prompt("Enter side b:", 0); //Convert to number using plus sign
let hypo = Math.sqrt(Math.pow(a,2) + Math.pow(b,2)); //Pythagorean formula
alert("The hypotenuse = " + hypo); //Browser output
console.log("The hypotenuse = " + hypo); //Console output
```

9.2 Data Types

- Date and Time
  - Core JavaScript includes Date() constructor for creating objects that represent dates and times
  - Date objects have methods that provide API for simple date computations
  - Date objects are not fundamental/primitive type like numbers are
  - Date object can be initialized in following four ways:
    - new Date() → current date and time
    - new Date(milliseconds) → number of milliseconds passed since reference date (01 Jan 1970)
    - new Date(datestring) → provide valid date string
    - new Date(year, month [, day, hours, minutes, seconds, ms]) → required year and month, first day and midnight is default
  - **Note:** Use Date() method with new keyword returns current date/time as string

9.2 Data Types

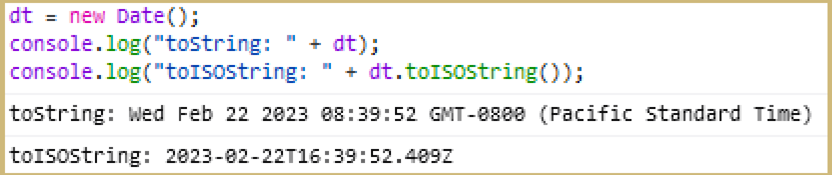
- Date and Time
  - Date/time methods
  - **<u>Note:</u>** getFullYear uses 4-digit years, old method getYear uses 2-digit year (deprecated, do not use anymore!)
  - **<u>Note:</u>** January is represented by 0 and December by 11!

Method	Description
getFullYear()	Get year as a four digit number (yyyy)
getMonth()	Get month as a number (0-11)
getDate()	Get day as a number (1-31)
getDay()	Get weekday as a number (0-6)
getHours()	Get hour (0-23)
getMinutes()	Get minute (0-59)
getSeconds()	Get second (0-59)
getMilliseconds()	Get millisecond (0-999)
getTime()	Get time (milliseconds since January 1, 1970)

  - JavaScript does not have built-in methods to perform date/time arithmetic
  - Need to write your own functions or download packages

9.2 Data Types

- Displaying Dates:
  - JavaScript will (by default) output dates in full text string format (using toString() method)
  - toISOString() method returns string in simplified extended ISO format (ISO 8601):
    - Always 24 or 27 characters long :
      - YYYY-MM-DDTHH:mm:ss.sssZ
      - Or ±YYYYYY-MM-DDTHH:mm:ss.sssZ (expanded year)
  - Timezone is always zero UTC offset (Denoted by suffix Z)



```
dt = new Date();
console.log("toString: " + dt);
console.log("toISOString: " + dt.toISOString());
```

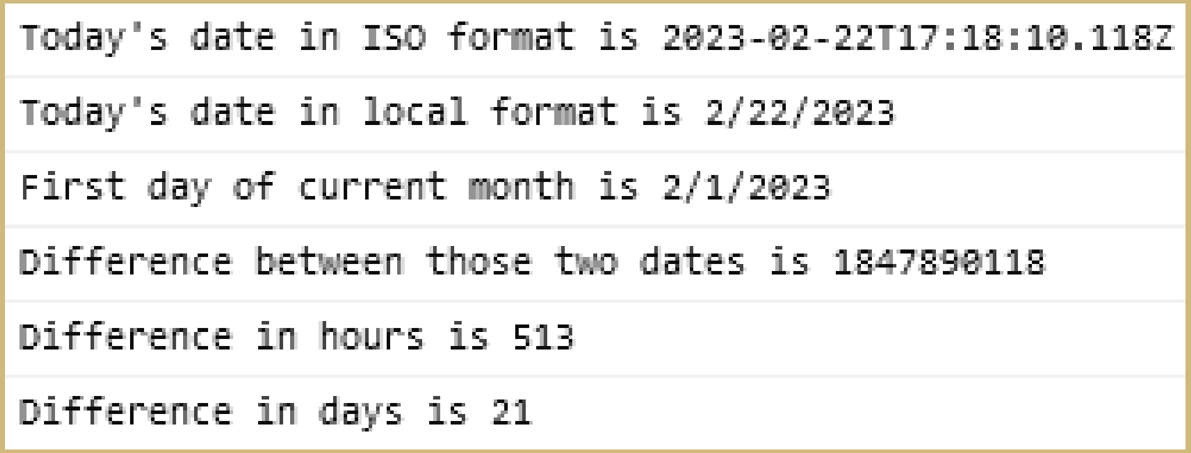
```
toString: Wed Feb 22 2023 08:39:52 GMT-0800 (Pacific Standard Time)
toISOString: 2023-02-22T16:39:52.409Z
```

Date Format Method	Description
toDateString()	Converts the date portion of a Date object into a readable string
toGMTString()	Deprecated. Use the toUTCString() method instead
toISOString()	Returns the date as a string, using the ISO standard
toJSON()	Returns the date as a string, formatted as a JSON date
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
UTC()	Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time

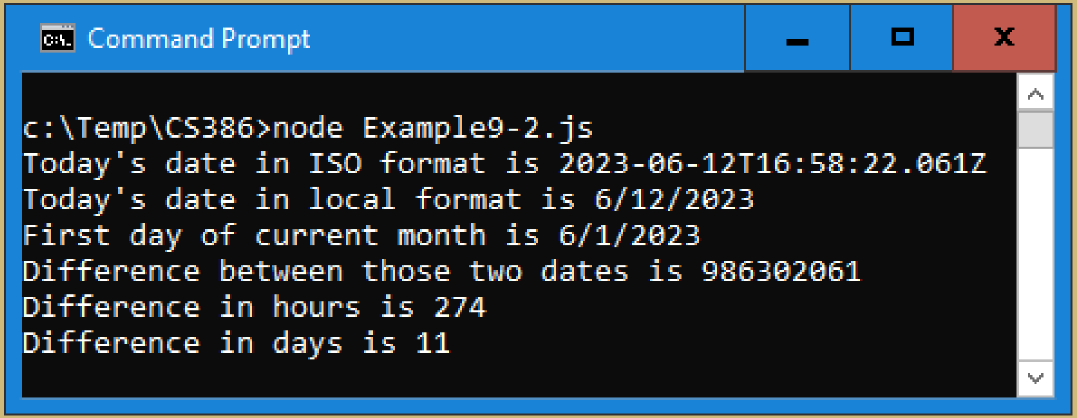
9.2 Data Types

➢ **Example 9-2:**

- ❑ Create variable dt, assign it current date/time object
- ❑ Create variable firstDayMonth, assign it first day of current month and year using dt
- ❑ Display today's date in ISO format
- ❑ Displays today's date in local date format only
- ❑ Create variable diff, assign difference current date and first of month (simply subtract date objects)
- ❑ Display difference in milliseconds (default), hours and days



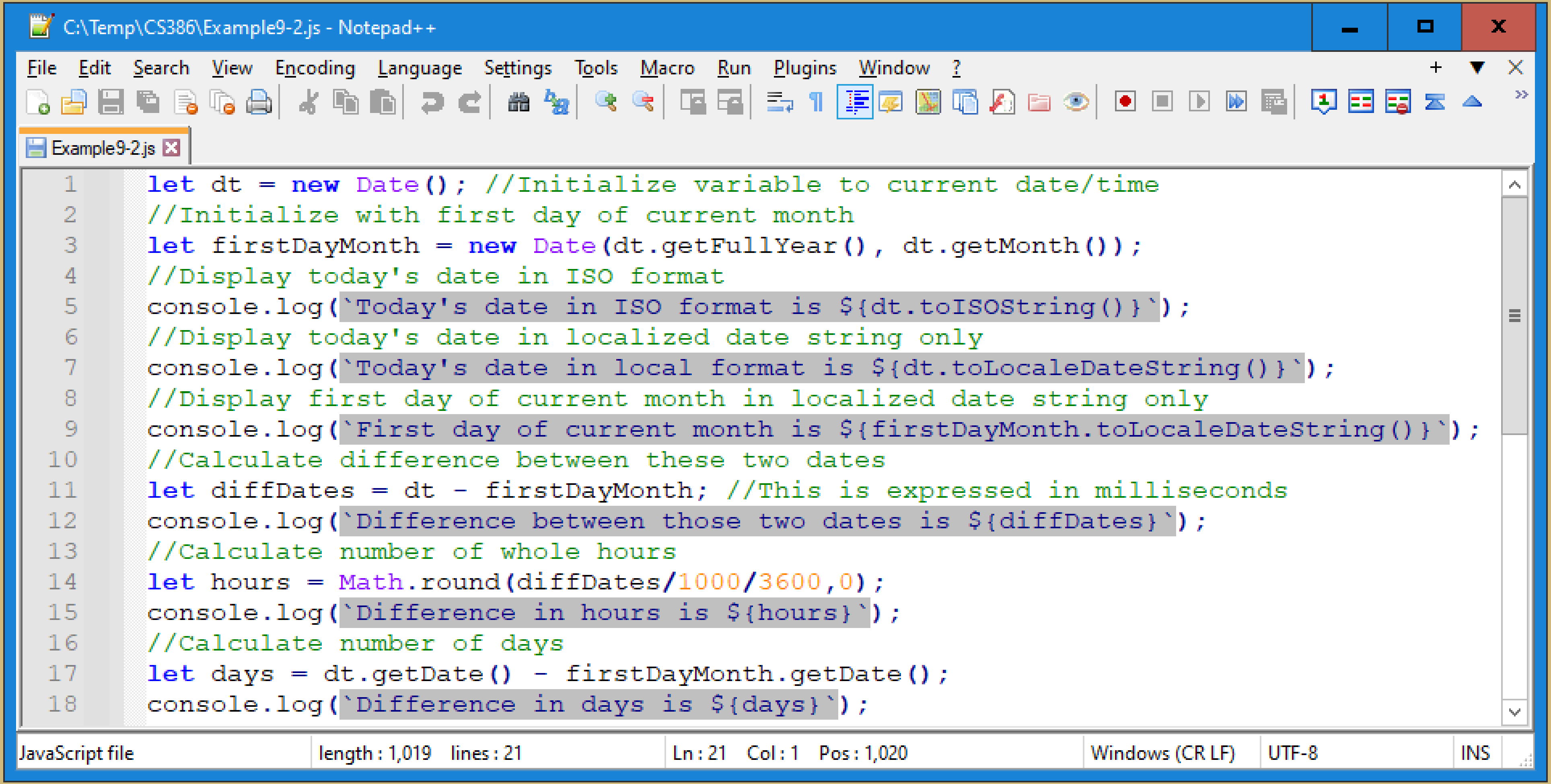
```
Today's date in ISO format is 2023-02-22T17:18:10.118Z
Today's date in local format is 2/22/2023
First day of current month is 2/1/2023
Difference between those two dates is 1847890118
Difference in hours is 513
Difference in days is 21
```



```
c:\Temp\CS386>node Example9-2.js
Today's date in ISO format is 2023-06-12T16:58:22.061Z
Today's date in local format is 6/12/2023
First day of current month is 6/1/2023
Difference between those two dates is 986302061
Difference in hours is 274
Difference in days is 11
```

9.2 Data Types

- **Example 9-2:**


C:\Temp\CS386\Example9-2.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example9-2.js

```javascript
let dt = new Date(); //Initialize variable to current date/time
//Initialize with first day of current month
let firstDayMonth = new Date(dt.getFullYear(), dt.getMonth());
//Display today's date in ISO format
console.log(`Today's date in ISO format is ${dt.toISOString()}`);
//Display today's date in localized date string only
console.log(`Today's date in local format is ${dt.toLocaleDateString()}`);
//Display first day of current month in localized date string only
console.log(`First day of current month is ${firstDayMonth.toLocaleDateString()}`);
//Calculate difference between these two dates
let diffDates = dt - firstDayMonth; //This is expressed in milliseconds
console.log(`Difference between those two dates is ${diffDates}`);
//Calculate number of whole hours
let hours = Math.round(diffDates/1000/3600,0);
console.log(`Difference in hours is ${hours}`);
//Calculate number of days
let days = dt.getDate() - firstDayMonth.getDate();
console.log(`Difference in days is ${days}`);
```

9.2 Data Types

- Text
  - To include string literally in JavaScript, simply enclose characters of string within matched pair of single or double quotes ('or ")
  - **Note:** When using single quotes for apostrophes or contractions, enclose string in double quotes
  - Backslash character (\) has special purpose in JavaScript strings → Escape characters:
    - Combined with character that follows it, represents character that is not otherwise representable within string → \n newline or line break
    - \' allows to escape from usual interpretation of single-quote character, which is to mark end of string

9.2 Data Types

- Text
  - To concatenate strings, use + operator (also arithmetic addition!)
  - <u>Example:</u>
    - "Hello" + " " + "World"
  - Determine length of string use length property of string:
  - Many other string methods
  - Remember: String is not object but behaves like one as it does have built-in methods

<u>Syntax:</u>
*string***.length**

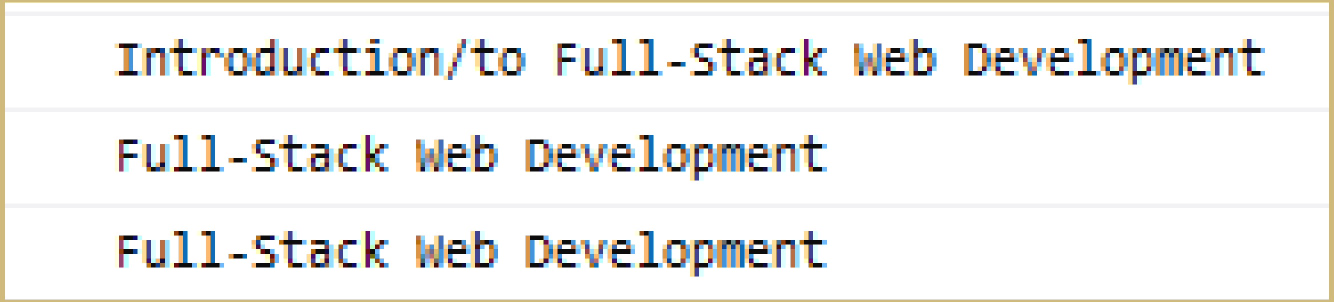
9.2 Data Types

- Some common string methods

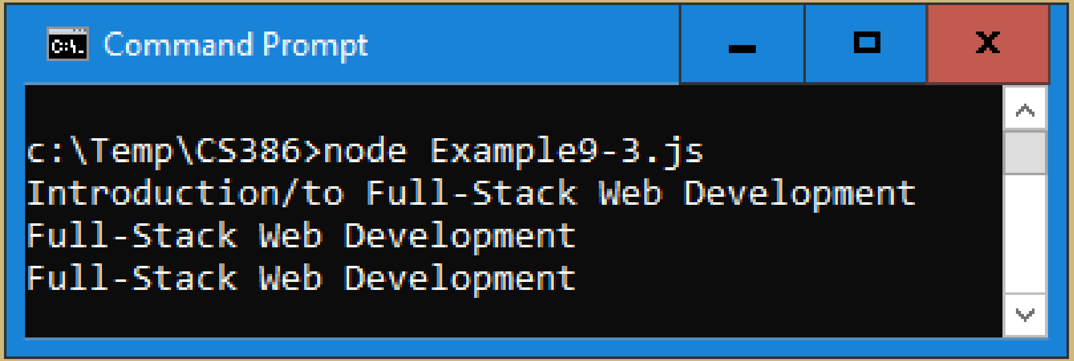
Method	Description
charAt(position)	Returns the character at the specified position (in Number). First position is 0 and last position is length - 1
indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.
lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.
replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string (only first occurrence). Regular expression can also be used as searchValue to replace multiple occurrences.
slice(startNumber [, endNumber])	Extracts a section of a string based on specified starting and ending index and returns a new string. endNumber is optional and by default is set to length of the string.
split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, [length]) (deprecated)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start [, end])	Returns the characters in a string between start and end indexes. End is optional and by default is set to the end of the string.
toLowerCase()	Returns lower case string value.
toString()	Returns the value of String object.
toUpperCase()	Returns upper case string value.

9.2 Data Types

- **Example 9-3:**
- Create variable string and assign “Introduction to Full-Stack Web Development”
- Replace first blank space with forward slash, output to console
- Extract substring “Full-Stack Web Development”, output to console
- Use substring and slice method



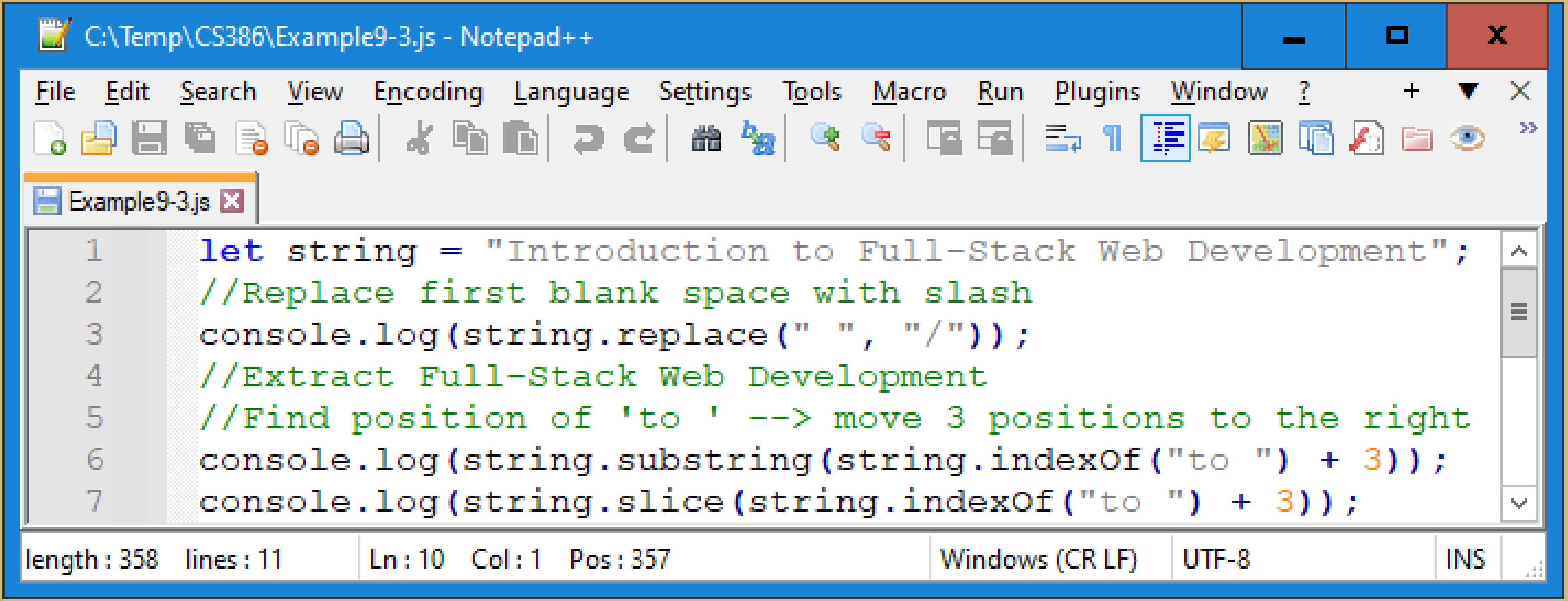
```
Introduction/to Full-Stack Web Development
Full-Stack Web Development
Full-Stack Web Development
```



```
c:\Temp\CS386>node Example9-3.js
Introduction/to Full-Stack Web Development
Full-Stack Web Development
Full-Stack Web Development
```

9.2 Data Types

- **Example 9-3:**


C:\Temp\CS386\Example9-3.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example9-3.js

```
let string = "Introduction to Full-Stack Web Development";
//Replace first blank space with slash
console.log(string.replace(" ", "/"));
//Extract Full-Stack Web Development
//Find position of 'to ' --> move 3 positions to the right
console.log(string.substring(string.indexOf("to ") + 3));
console.log(string.slice(string.indexOf("to ") + 3));
```

length : 358 lines : 11 Ln : 10 Col : 1 Pos : 357 Windows (CR LF) UTF-8 INS

9.2 Data Types

- Boolean Values
  - True and False
  - In JavaScript, every value has Boolean value
  - Following values are false:
    - undefined
    - null
    - 0
    - -0
    - NaN
    - "" // empty string
  - Every other value is true, even empty object ({ }), for example
  - Boolean operators:
    - AND: &&
    - OR: ||
    - NOT: !

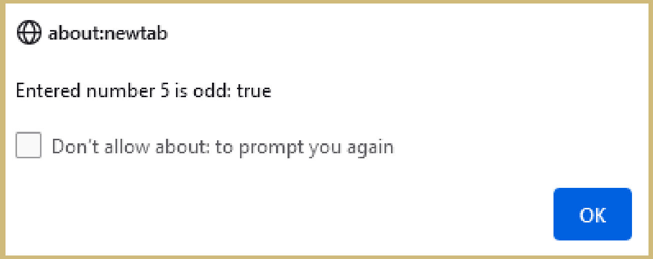
9.2 Data Types

- Boolean Conversion Function
  - To convert any value into Boolean true or false
  - Examples:
    - Boolean(0) → returns false
    - Boolean(23) → returns true
  - Remember:
    - Few values inherently false, such as 0, -0, empty string, null, undefined, NaN
    - Every other value including empty objects are true
  - Exploit this logic in your code

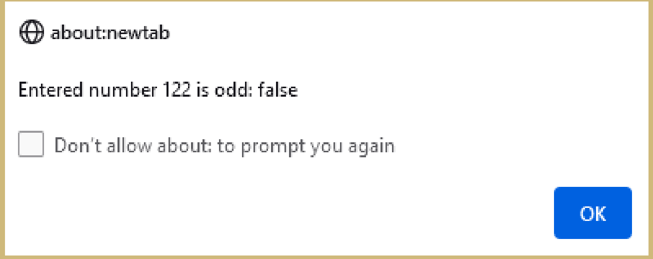
Syntax:
Boolean(*value*)

9.2 Data Types

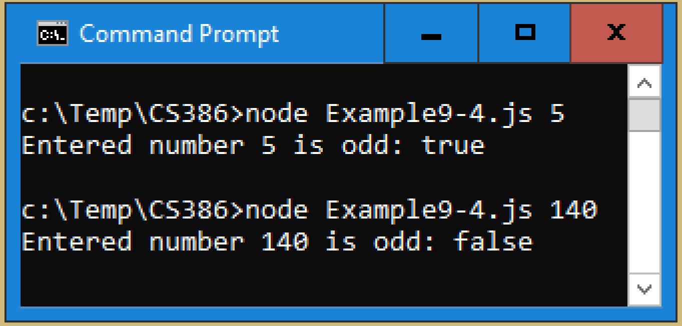
- **Example 9-4:**
- Program to determine whether entered number is even or odd
  - ❑ Declare variables num and result
  - ❑ Use try..catch for input (prompt in browser, argument vector in node.js):
    - o Store entered number in num
  - ❑ Divide variable num by 2 using modulo operator and store in variable result:
    - o If remainder is 0 → even, if not 0 → odd
  - ❑ Use try..catch for output (alert in browser, console.log in node.js):
    - o Use Boolean function to convert variable result into true or false
    - o Produce following output:


about:newtab

Entered number 5 is odd: true


about:newtab

Entered number 122 is odd: false

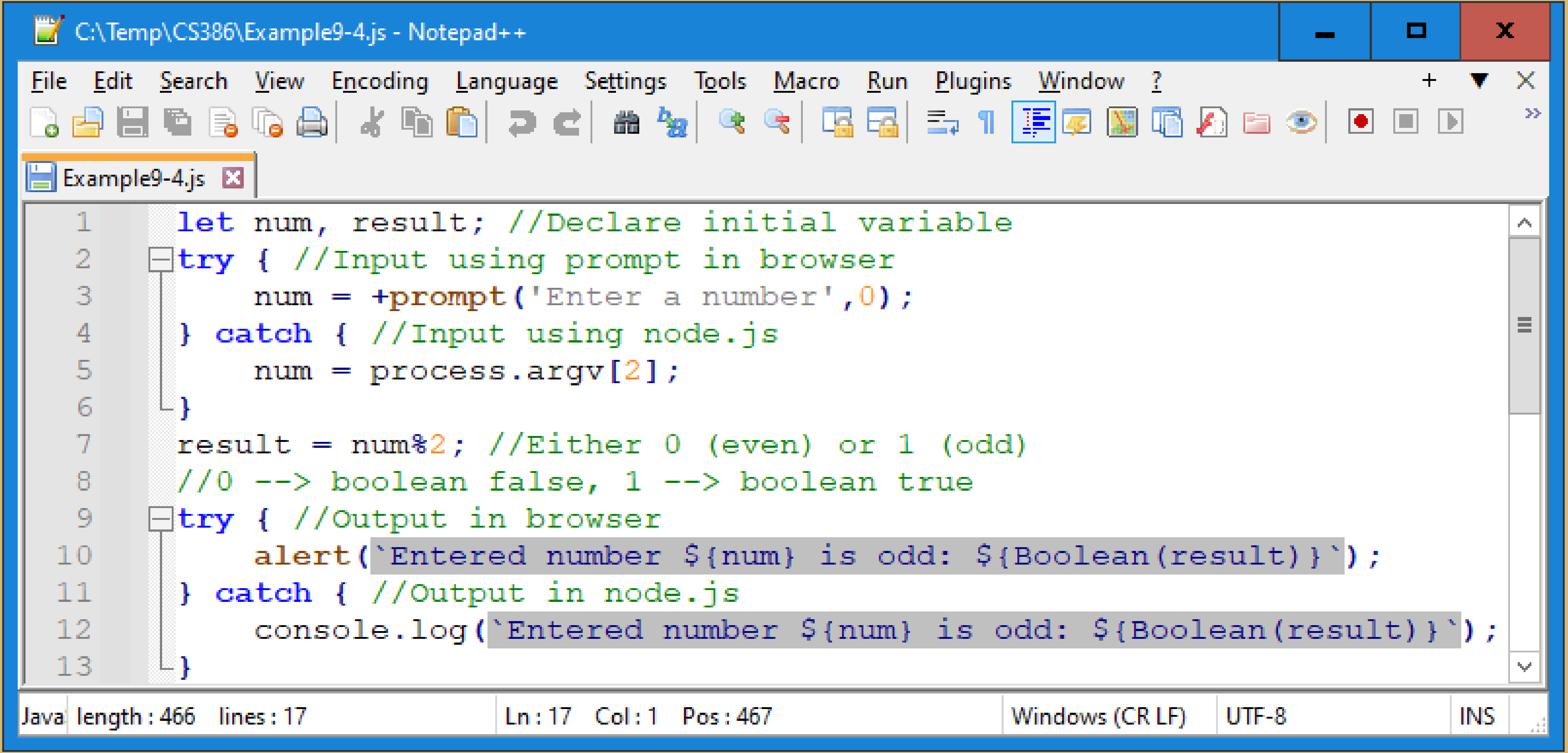


```
c:\Temp\CS386>node Example9-4.js 5
Entered number 5 is odd: true

c:\Temp\CS386>node Example9-4.js 140
Entered number 140 is odd: false
```

9.2 Data Types

- **Example 9-4:**


C:\Temp\CS386\Example9-4.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example9-4.js

```
let num, result; //Declare initial variable
try { //Input using prompt in browser
    num = +prompt('Enter a number',0);
} catch { //Input using node.js
    num = process.argv[2];
}
result = num%2; //Either 0 (even) or 1 (odd)
//0 --> boolean false, 1 --> boolean true
try { //Output in browser
    alert(`Entered number ${num} is odd: ${Boolean(result)}`);
} catch { //Output in node.js
    console.log(`Entered number ${num} is odd: ${Boolean(result)}`);
}
```

Java length : 466 lines : 17 Ln : 17 Col : 1 Pos : 467 Windows (CR LF) UTF-8 INS

9.3 Values and Conversion

- JavaScript is dynamically typed!
- Variables are declared without binding type, value assigned determines type
- Other languages require type declaration when declaring variables
- Statically Typed:
  - Type is bound to variable
  - Types are checked at compile time (compiler or interpreter)
- Dynamically Typed:
  - Type is bound to value
  - Types are checked at run time

9.3 Values and Conversion

- When combining different types using operators, some languages perform implicit conversions while others do not and throw errors
- Strongly Typed:
  - Variable will not be automatically converted from one type to another
  - Python (Dynamically typed): s = "abc" + 123 # Type Error
- Weakly Typed:
  - Variables can be implicitly converted from one type to another
  - Java (Statically typed): String s = "abc" + 123; // "abc123"
  - JavaScript (Dynamically typed): let s = "abc" + 123; //"abc123"
- Various languages:
  - JavaScript is dynamically typed and weakly typed!!
  - Java is statically typed and weakly typed
  - Python is dynamically typed and strongly typed
  - C# is statically typed and strongly typed

9.3 Values and Conversion

- Implicit Conversion
  - Because JavaScript is weakly typed, implicit conversion takes place
  - Can lead to unexpected results
  - Example:
    - "7" * "4" = 28
    - 7 + "4" = "74"
  - In general, explicit type conversion is better
  - Developer has exact control over conversion rather than JavaScript (implicit)
  - Simple equality operator (==) also performs implicit conversion before comparing operands:
    - null == undefined // These two values are treated as equal
    - "0" == 0 // String converts to number before comparing
    - 0 == false // Boolean converts to number before comparing
    - "0" == false // Both operands convert to numbers before comparing
  - Strict equality operator (===) does not perform type conversion
    - Both operands must be of same type, otherwise comparison is already false

9.3 Values and Conversion

- Explicit Conversion
  - Simplest way to perform explicit type conversion is to use type conversion functions:
    - Boolean()
    - Number()
    - String()
    - Object()
  - Formatting and parsing numbers are common tasks in computer programs
  - JavaScript has specialized functions and methods that provide more precise control over number-to-string and string-to-number conversions
  - toString() method defined by Number class accepts optional argument that specifies radix, or base, for conversion
  - <u>Example:</u>
    - let n = 17;
    - n.toString(2); // Evaluates to "10001"
    - "0" + n.toString(8); // Evaluates to "021"
    - "0x" + n.toString(16); // Evaluates to "0x11"

9.3 Values and Conversion

- Explicit Conversion
  - Number class defines three methods for of number to-string conversions:
    - toFixed() converts number to string with specified number of digits after decimal point (It never uses exponential notation)
    - toExponential() converts number to string using exponential notation, with one digit before decimal point and specified number of digits after decimal point
    - toPrecision() converts number to string with number of significant digits you specify using exponential notation if number of significant digits is not large enough to display entire integer portion of number

Syntax:
num.**toFixed(***d***)**
d = number of decimals (optional, default 0)

Syntax:
num.**toExponential(***d***)**
d = number of decimals (optional, sets as many as possible)

Syntax:
num.**toPrecision(***d***)**
d = number of decimals (optional, number is returned without any formatting)

9.3 Values and Conversion

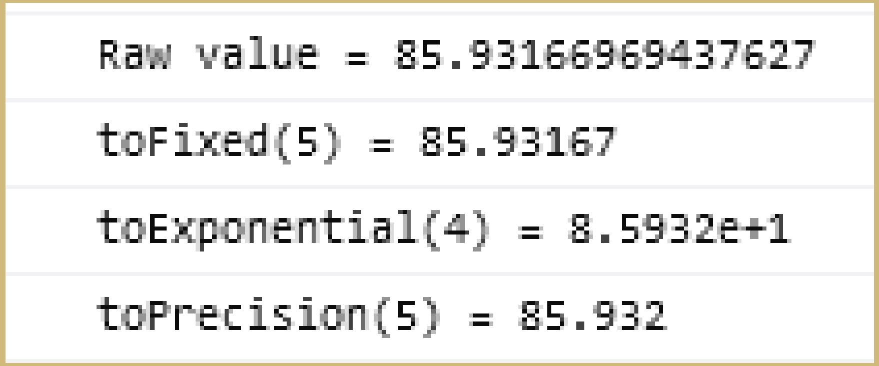
- Explicit Conversion
  - parseInt() and parseFloat() functions (global functions, not methods of any class) are more flexible:
    - parseInt() parses only integers
    - parseFloat() parses both integers and floating-point numbers
    - If string begins with "0x" or "0X", parseInt() interprets it as hexadecimal number
  - Additional rules for parseInt() and parseFloat() methods:
    - Skip leading whitespace
    - Parse as many numeric characters as they can
    - Ignore anything that follows
    - If first nonspace character is not part of valid numeric literal, they return NaN
    - parseInt() accepts optional second argument specifying radix (base) of number to be parsed (Legal values are between 2 and 36)

Syntax:
parseInt(*string [, radix]***)**

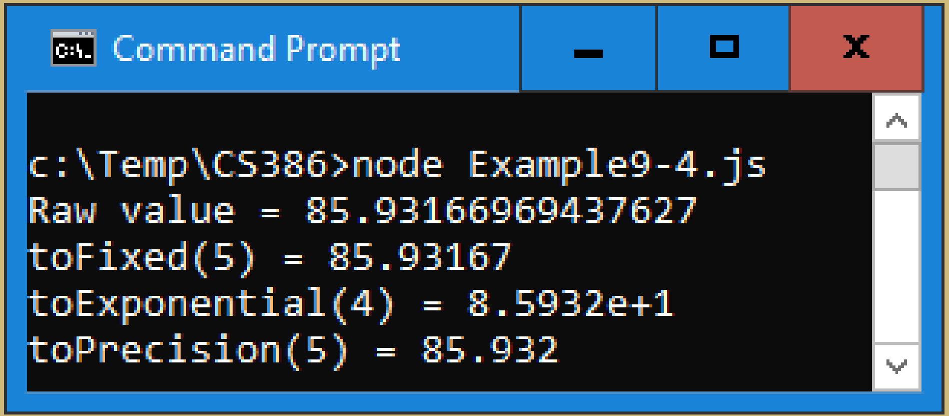
Syntax:
parseFloat(*string***)**

9.3 Values and Conversion

- **<u>Example 9-5:</u>**
- Create variable radius and assign 5.23 to it
- Create variable circleArea and assign area of circle (π * radius^2)
- Math library to use PI and pow (power) functions
- First show raw (unformatted) area of circle
- Then use number formatting functions:



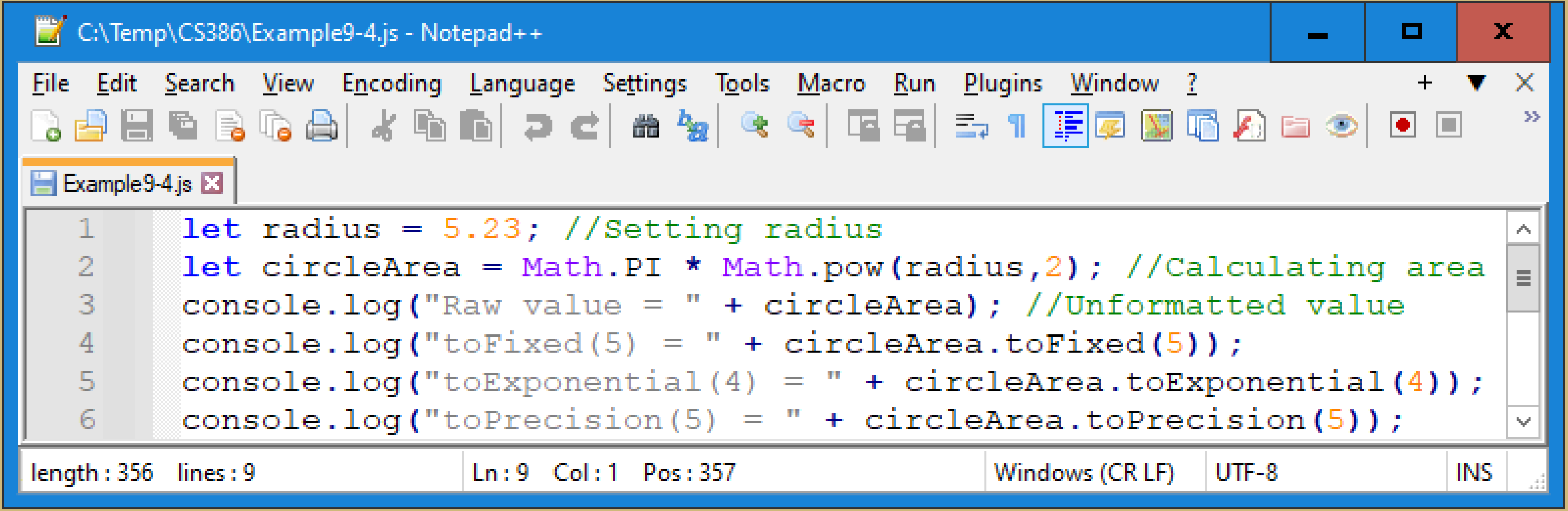
```
Raw value = 85.93166969437627
toFixed(5) = 85.93167
toExponential(4) = 8.5932e+1
toPrecision(5) = 85.932
```



```
c:\Temp\CS386>node Example9-4.js
Raw value = 85.93166969437627
toFixed(5) = 85.93167
toExponential(4) = 8.5932e+1
toPrecision(5) = 85.932
```

9.3 Values and Conversion

➢ **Example 9-5:**


C:\Temp\CS386\Example9-4.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example9-4.js

```javascript
let radius = 5.23; //Setting radius
let circleArea = Math.PI * Math.pow(radius,2); //Calculating area
console.log("Raw value = " + circleArea); //Unformatted value
console.log("toFixed(5) = " + circleArea.toFixed(5));
console.log("toExponential(4) = " + circleArea.toExponential(4));
console.log("toPrecision(5) = " + circleArea.toPrecision(5));
```

length : 356 lines : 9 Ln : 9 Col : 1 Pos : 357 Windows (CR LF) UTF-8 INS