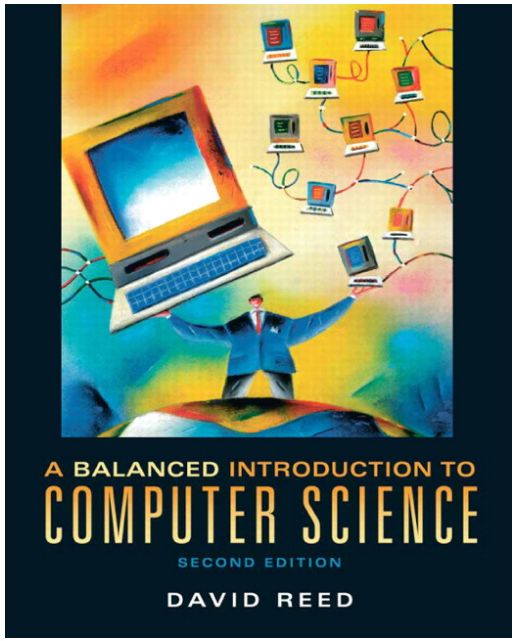
A Balanced Introduction to Computer Science, 2/E

David Reed, Creighton University

ISBN 978-0-13-601722-6



Chapter 5
JavaScript Numbers and Expressions

Data Types

each unit of information processed by a computer belongs to a general category or *data type*

- e.g., string, number, Boolean (either true or false)

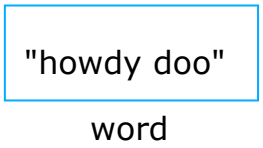
each data type is associated with a specific set of predefined operators that may be used by programmers to manipulate values of that type

- e.g., we have seen string concatenation via +
- similarly, standard operators are predefined for numbers
  - addition (+), subtraction (-), multiplication (*), division (/)

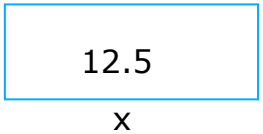
variables can be assigned various kinds of numerical values, including mathematical expressions formed by applying operators to numbers

- when an expression appears on the right-hand side, the expression is evaluated and the resulting value is assigned to the variable on the left-hand side

```
word = "howdy" + " doo";
```




```
x = 50/4;
```



Variables and Expressions

similarly, expressions can appear in write statements

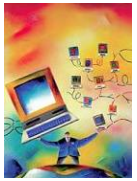
- note: parentheses can be used to make sub-expression grouping explicit

```
document.write(3 + 7);
```
➔ writes `10`

```
document.write("The sum of is " + (3 + 7));
```
➔ writes `The sum is 10`

if a variable appears in an expression, the value currently assigned to that variable is substituted

Number Representation



useful facts about JavaScript numbers

- to improve readability, very large or very small number are displayed in *scientific notation:* `XeY` represents the value $X \times 10^Y$
  - e.g., `1e24` → 1×10^{24} → `100000000000000000000000`

- JavaScript stores all numbers in memory cells of a fixed size (64 bits)
  - as a result, only a finite number of values can be represented
  - e.g., `1e308` can be represented, but `1e309` is treated as `Infinity`
    `1e-323` can be represented, but `1e-324` is treated as `0`

- even within the range 1e-323 . . . 1e309, not all numbers can be represented
  - note that between any two numbers lie infinitely more numbers!
  - JavaScript can represent approximately 17 significant digits
  - e.g.., `0.9999999999999999` can be represented exactly
    `0.99999999999999999` is rounded up to `1`

Mixed Expressions

in JavaScript, the + operator serves two purposes

- when applied to numbers, + means addition
- when applied to strings, + means concatenation
- what about a mixed expression?

when applied to a string and a number,

- the number is converted to a string (effectively, by placing quotes around it),
- then string concatenation is performed

```
"We're number " + 1  →  "We're number " + "1"
                     →  "We're number 1"
```

note: expressions involving + are evaluated left-to-right

- this can have consequences in the way mixed expressions are evaluated
- *ADVICE: always use parentheses to group nested sub-expressions*

```
3 + 2 + " is the sum"   →  (3 + 2) + " is the sum"
                        →  5 + " is the sum"
                        →  "5" + " is the sum"
                        →  "5 is the sum"
"the sum is " + 3 + 2   →  ("the sum is " + 3) + 2
                        →  ("the sum is " + "3") + 2
                        →  "the sum is 3" + 2
                        →  "the sum is 3" + "2"
                        →  "the sum is 32"
```

Prompting for Numbers

special care must be taken when prompting the user for number values

- recall that `prompt` always returns a string, even if the user enters only digits
- e.g., if the user enters 500 at a prompt, then the value "500" is returned

```
myNumber = prompt("Enter a number", "");
document.write("One more is " + (myNumber + 1));
```

- if the user entered 12 at the prompt, what would be displayed?
- the message displayed would be `One more is 121` WHY?
  - the prompt returns `"12"` which is stored in `myNumber`
  - the parenthesized sub-expression `(myNumber + 1)` is evaluated first
  - since this is a mixed expression, the number `1` is converted to `"1"` then concatenated
  - the result, `"121"`, is then concatenated to the end of `"One more is "`

what is needed is a mechanism for converting strings of digits into numbers

- e.g., "500" → 500, "1.314" → 1.314, …
- this is accomplished in JavaScript using the `parseFloat` *function*

Functions

in mathematics, a *function* is a mapping from inputs to a single output

- e.g., the absolute value function maps one number to another
  -5 → 5, -2.4 → 2.4, 17 → 17, …

$$|n| = \begin{cases} n & \text{if } n \geq 0 \\ -n & \text{if } n < 0 \end{cases}$$

- similarly, the `parseFloat` function maps strings of digits to numbers
  "500" → 500, "1.314" → 1.314, "0" → 0, …

from a programmer's view, a function is a "unit of computational abstraction"

- there is some computation required to calculate the output given the input(s)
- a JavaScript function encapsulates that computation and hides the details
- the user does not need to know how the function works, only how to apply it
  - applying a function to inputs is known as *calling the function*
  - the output of a function call is known as the *return value*

parseFloat

a function call can appear anywhere in a JavaScript expression

- when the expression is evaluated, the return value for that call is substituted

```
myNumber = prompt("Enter a number", "");
myNumber = parseFloat(myNumber);
document.write("One more is " + (myNumber + 1));
```

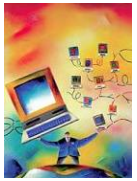
- the 1st statement prompts the user and stores their input (say "12")in myNumber
- the 2nd statement calls parseFloat to convert the string to a number (12) and then reassigns that number back to myNumber
- the 3rd statement uses the number value 12 to display One more is 13

note, the following is not an error (but probably not what was intended)

```
myNumber = prompt("Enter a number", "");
parseFloat(myNumber);
document.write("One more is " + (myNumber + 1));
```

- the call to parseFloat returns a number, but nothing is done with that number
- NOTE: the only way to change the value of a variable is via an assignment statement

Temperature Conversion

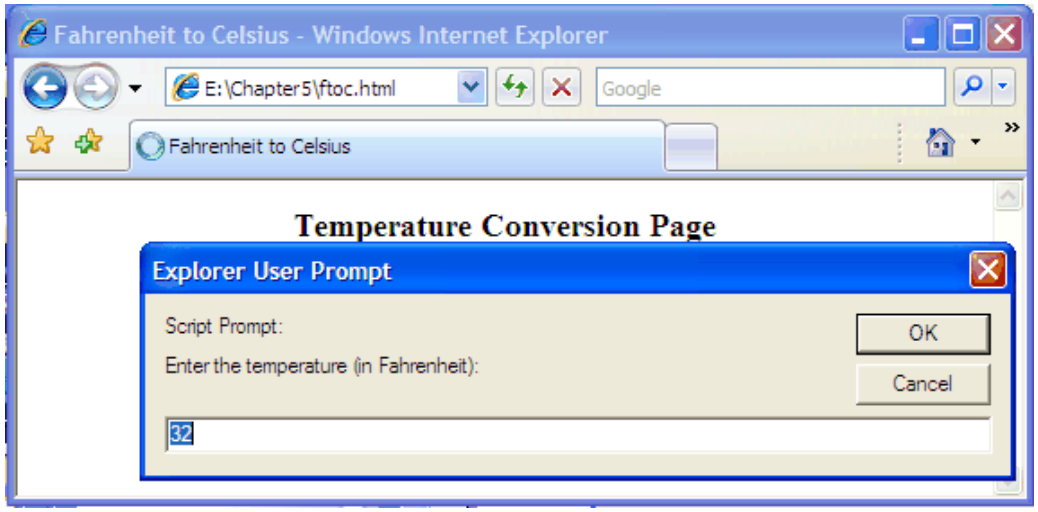


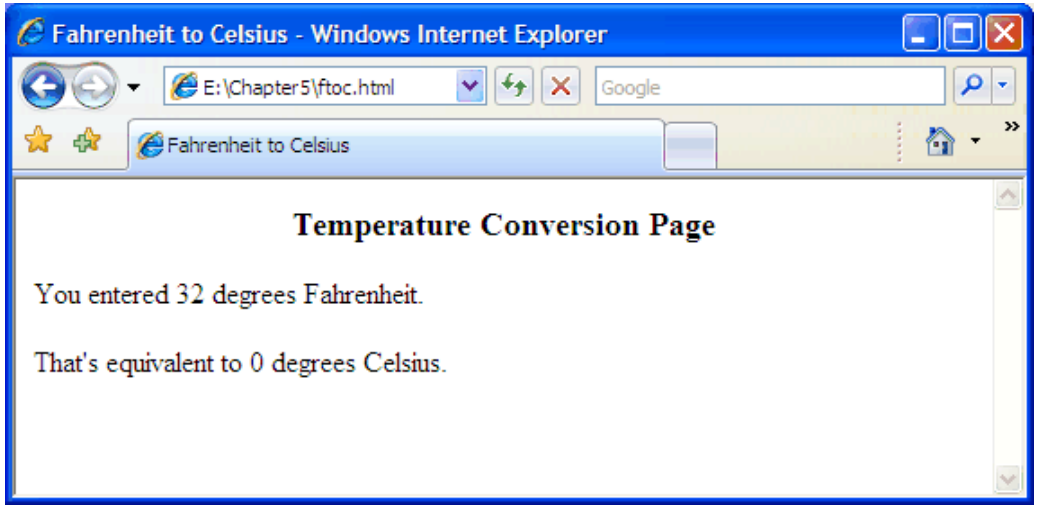
the following page prompts the user for a temperature (in Fahrenheit), stores the input as a number, then converts that temperature to Celsius

```
 1. <html>
 2.   <!-- ftoc.html                                        Dave Reed -->
 3.   <!-- Converts a temperature from Fahrenheit to Celsius.   -->
 4.   <!-- ================================================== -->
 5.
 6.   <head>
 7.     <title>Fahrenheit to Celsius</title>
 8.   </head>
 9.
10.   <body>
11.     <h3 style="text-align:center">Temperature Conversion Page</h3>
12.
13.     <script type="text/javascript">
14.       tempInFahr = prompt("Enter the temperature (in Fahrenheit):", "32");
15.       tempInFahr = parseFloat(tempInFahr);
16.
17.       tempInCelsius = (5/9) * (tempInFahr - 32);
18.
19.       document.write("<p>You entered " + tempInFahr + " degrees Fahrenheit.</p>");
20.       document.write("<p>That's equivalent to " + tempInCelsius +
21.                      " degrees Celsius.</p>");
22.     </script>
23.   </body>
24. </html>
```

Conversion Page

note that the prompt
has a default value of 32





Common Pattern

many tasks that we will consider have the same basic form

1. prompt the user for numbers
2. store them in variables
3. perform some calculation(s) using those numbers
4. display the results

not surprisingly, there is a pattern to the code

```
<script type="text/javascript">
  number1 = prompt("PROMPT MESSAGE", "");
  number1 = parseFloat(number1);
  number2 = prompt("PROMPT MESSAGE", "");
  number2 = parseFloat(number2);
  . . .
  numberN = prompt("PROMPT MESSAGE", "");
  numberN = parseFloat(numberN);

  answer = SOME EXPRESSION INVOLVING number1, …, numberN;

  document.write("MESSAGE INVOLVING answer");
</script>
```

Predefined Functions

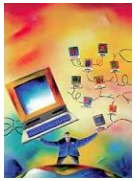
JavaScript provides an extensive library of predefined mathematical functions

- `Math.sqrt` returns the square root of a number
  e.g., `Math.sqrt(9)` → `3`

- `Math.max` returns the maximum of two numbers
  e.g., `Math.max(3.2, 1.8)` → `3.2`

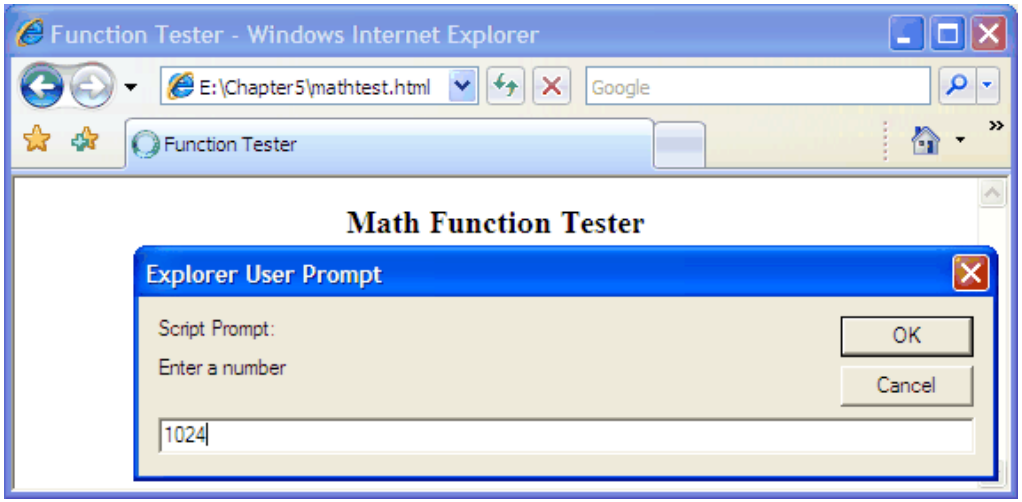
```
1.  <html>
2.    <!-- mathtest.html                    Dave Reed -->
3.    <!-- This page tests the Math.sqrt function.  -->
4.    <!-- ========================================= -->
5.
6.    <head>
7.      <title>Function Tester</title>
8.    </head>
9.
10.   <body>
11.     <h3 style="text-align:center">Math Function Tester</h3>
12.
13.     <script type="text/javascript">
14.       number = prompt("Enter a number", 0);
15.       number = parseFloat(number);
16.
17.       result = Math.sqrt(number);
18.       document.write("<p>Math.sqrt(" + number + ") = " + result + "</p>");
19.     </script>
20.   </body>
21. </html>
```

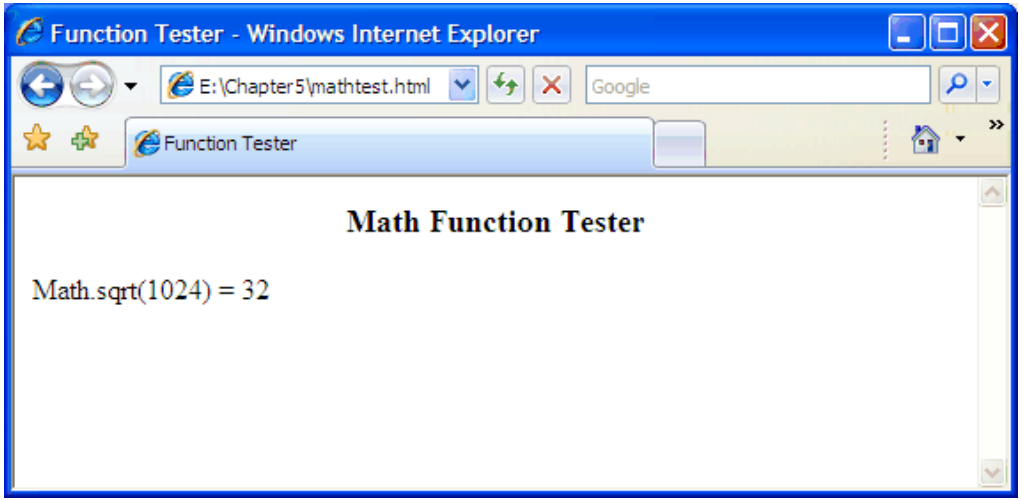
Tester Page



this page could be modified to test a variety of functions

- change the function call in the page
- enter various inputs and observe the corresponding outputs





Other Useful Functions

`Math.pow` raises a number to a power

```
Math.pow(2, 10)      → 2^10 = 1024
Math.pow(2, -1)      → 2^-1 = 0.5
Math.pow(9, 0.5)     → 9^0.5 = 3
```

`Math.random` generates a random number in the range [0…1)

- note: this function has no inputs; it returns a different number each call

```
Math.random()        → 0.330085256267488 14
Math.random()        → 0.213335955823927
Math.random()        → 0.8975001737758223
.
.
.
```

Errors and Debugging

in computer jargon, the term *bug* refers to an error in a program

- the process of systematically locating and fixing errors is *debugging*

three types of errors can occur

1. *syntax errors:* typographic errors
   - e.g., omitting a quote or misspelling a function name
   - since the browser catches these, they are usually "easy" to identify and fix
2. *run-time errors:* occur when operations are applied to illegal values
   - e.g., attempting to multiply a string or divide by zero
   - also caught by the browser, which either produces an error message or else returns a special value (string multiplication produces `NaN`, for "`Not a Number`"; division by zero produces `Infinity`)
3. *logic errors:* flaws in the design or implementation of a program
   - whenever your program produces the wrong result
   - since they are not caught by the browser (the program is legal, just not what you wanted), logic errors are hardest to identify

useful technique for identifying bugs: *diagnostic write statements*

- at various intervals in the code, write out the values of key variables
- you can then isolate at what point the program is going wrong