What Are Variables in JavaScript?

- Variables in JavaScript have the same meaning as variables in algebra. That is, they represent some unknown, or variable, value.

\[
x = a + b
\]
\[
z + 2 = 3(y - 5)
\]

- Remember that variables in algebra are represented by a single alphabetic character.
- They are "containers" that hold values.

Legal Identifiers in JavaScript

- Another name for a variable in JavaScript is an identifier.
- Variables in JavaScript may be given representations containing multiple characters. But there are rules for these representations.
- Legal variable names in JavaScript:
  - May only consist of letters, digits, and underscores
  - Can not have blank spaces
  - May not begin with a number
  - May not be a JavaScript reserved word (keyword).

Reserved Words (Keywords) in JavaScript

<table>
<thead>
<tr>
<th>abstract</th>
<th>delete</th>
<th>function</th>
<th>null</th>
<th>throw</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>do</td>
<td>goto</td>
<td>package</td>
<td>throws</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>if</td>
<td>private</td>
<td>transient</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>implements</td>
<td>protected</td>
<td>true</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>import</td>
<td>public</td>
<td>try</td>
</tr>
<tr>
<td>catch</td>
<td>export</td>
<td>in</td>
<td>return</td>
<td>typeof</td>
</tr>
<tr>
<td>char</td>
<td>extends</td>
<td>instanceof</td>
<td>short</td>
<td>var</td>
</tr>
<tr>
<td>class</td>
<td>false</td>
<td>int</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>const</td>
<td>final</td>
<td>interface</td>
<td>super</td>
<td>volatile</td>
</tr>
<tr>
<td>continue</td>
<td>finally</td>
<td>long</td>
<td>new</td>
<td>switch</td>
</tr>
<tr>
<td>debugger</td>
<td>float</td>
<td>native</td>
<td>synchronized</td>
<td>with</td>
</tr>
<tr>
<td>default</td>
<td>for</td>
<td>new</td>
<td>this</td>
<td></td>
</tr>
</tbody>
</table>

CMSC104 Naming Conventions

- For this class (and some future CS classes), we're going to use the following rules when naming variables:
  - Begin variable names with lowercase letters
  - Use meaningful names
  - Separate "words" within identifiers with underscores or mixed upper and lower case.
  - Examples: surfaceArea, surface_Area, surface_area
  - Be consistent!
**Case Sensitivity**

- JavaScript is **case sensitive**
  - It matters whether an identifier, such as a variable name, is uppercase or lowercase.
  - Example:
    - area
    - Area
    - AREA
    - ArEa
    - all seen as different variables.

**Legal Identifiers vs. Naming Conventions**

- **Legal identifiers** refer to the restrictions JavaScript places on naming identifiers, i.e., variable names cannot begin with a number.
- **Naming conventions** refer to the standards you must follow for this course, i.e., all variable names must begin with lowercase.

**Which Are Legal Identifiers?**

- AREA
- 3D
- lucky***
- Last-Chance
- x_yt3
- num+
- area_under_the_curve

**Which follow the CMSC104 Naming Conventions?**

- Area
- person1
- Last_Chance
- values
- x_yt3
- pi
- finaltotal
- numChildren
- area_under_the_curve

**Declaring Variables**

- Before using a variable, you need to **declare** it.
- The **declaration statement** includes the `var` keyword and the name of the variable.
- Examples of variable declarations:
  - `var meatballs;`
  - `var meatballs, area;`
  - `var area;`

**Declaring Variables (con’t)**

- When we declare a variable
  - Space is set aside in memory to hold the value
  - That space is associated with the variable `name`
  - The initial value of the variable is **undefined** (it is not 0!)
- Visualization of the declaration
  - `var meatballs;`
  - `meatballs`
  - `name`
  - `undefined`
More About Variables

- In JavaScript variables can hold four basic types of values
  - Numbers
    - i.e. 40, 15.5, 700
  - Strings
    - i.e. "Hello, World!", "Linux is cool!"
  - Booleans
    - i.e. true, false
  - Null
    - i.e. null

Using Variables: Initialization

- Variables may be be given initial values, or **initialized**, when declared. Examples:
  - `var length = 7;`
  - `var diameter = 5.9;`
  - `var message = "Hello!";`
  - `var walletEmpty = true;`

Using Variables: Assignment

- Variables may have values assigned to them through the use of an **assignment statement**.
  - Such a statement uses the **assignment operator** `=`
  - This operator does not denote equality. It assigns the value of the righthand side of the statement (the **expression**) to the variable on the lefthand side.
  - Examples:
    - `diameter = 5.9;`
    - `area = length * width;`
  - Note that only single variables may appear on the lefthand side of the assignment operator.

Brian's Shopping Trip Revisited

**Problem:** Brian bought a belt for $9 and a shirt that cost 4 times as much as the belt. He then had $10. How much money did Brian have before he bought the belt and shirt?

- Shirt
- Belt

Pseudocode

Display "Enter the price of the first item: "
Read <item 1 price>
Display "Enter the multiplier: "
Read <multiplier>
Display "Enter the amount left after shopping: "
Read <amount left>
<item2 price> = <multiplier> X <item1 price>
<start amount> = <item1 price> + <item2 price> + <amount left>
Display "The starting amount was ", <start amount>
Example: Declarations and Assignments

```javascript
<script type = "text/javascript">
<!>
var item1Price, multiplier;
var amountLeft, item2Price;
var startAmount;
item1Price = 9;
multiplier = 4;
amountLeft = 10;
item2Price = multiplier * item1Price;
startAmount = item1Price + item2Price + amountLeft;
</script>
(continued on next slide)
```

Example: Declarations and Assignments

```javascript
document.write("The cost of item 1: ");
document.write(item1Price);
document.write("\n");
document.write("The multiplier: ");
document.write(multiplier);
document.write("\n");
document.write("The money we had left: ");
document.write(amountLeft);
document.write("\n");
document.write("The starting amount was: ");
document.write(startAmount);
</script>
```

Screenshot of Variables Example

![Screenshot of Variables Example](image)

Enhancing Our Example

- What is the problem with our solution?
- It produces the same results every time!
- Let’s also ask the user to enter the values for our variables, rather than “hard-coding” them in.

Getting User Input

- Use the prompt() function
  - Will display a pop-up window asking the user to enter data
- Examples:
  ```javascript
  name = prompt("What is your name?");
  payRate = prompt("Enter your pay rate: ");
  score = prompt("Please enter the score: ");
  ```

```
The prompt() function is equivalent to the Display/Read in pseudocode.
```

Screenshot of prompt() example

![Screenshot of prompt() example](image)
Enhanced Variables Example

```javascript
<script type = "text/javascript">
var item1Price, multiplier, amountLeft, item2Price, startAmount;

item1Price = prompt('Please enter the cost of the first item: ');  
item1Price = parseFloat(item1Price);  
multiplier = prompt('Please enter the multiplier: ');  
multiplier = parseFloat(multiplier);  
amountLeft = prompt('Please enter the amount left: ');  
amountLeft = parseFloat(amountLeft);  

item2Price = multiplier * item1Price;  
startAmount = item1Price + item2Price + amountLeft;

document.write("The cost of item 1: $" + item1Price);  
document.write("<br /> 
The multiplier: " + multiplier);  
document.write("<br /> 
The money we had left: $" + amountLeft);  
document.write("<br /> 
The starting amount was: $" + startAmount);  

</script>

Changes Made to Include User Input

- Instead of giving the variables explicit initialization values, as in:
  ```javascript
  item1Price = 9;
  multiplier = 4;
  amountLeft = 10;
  ```
  we used the following:
  ```javascript
  item1Price = prompt('Please enter the cost of the first item: ');  
  item1Price = parseFloat(item1Price);  
  multiplier = prompt('Please enter the multiplier: ');  
  multiplier = parseFloat(multiplier);  
  amountLeft = prompt('Please enter the amount left: ');  
  amountLeft = parseFloat(amountLeft);  
  ```

Screenshot of Enhanced Variables Example

![Screenshot of Enhanced Variables Example](image-url)
Final Screenshot of Enhanced Variables Example

Brian's Shopping Trip
The cost of the first item: $9
The multiplier: 4
The total we had left: $10
The change amount: $55

Try it!  http://userpages.umbc.edu/~dblock/variables2.html

Good Programming Practices

- Place a comment before each logical “chunk” of code describing what it does.
- Do not place a comment on the same line as code (with the exception of variable declarations).
- Use spaces around all arithmetic and assignment operators.
- Use blank lines to enhance readability.

Good Programming Practices

- Place a blank line between the last variable declaration and the first executable statement of the program.
- Indent the body of the program 2 to 3 spaces -- be consistent!

Arithmetic Operators in JavaScript

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>num1 + num2</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>initial - spent</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>radius * 2</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>sum / count</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>m % n</td>
</tr>
</tbody>
</table>

Modulus

- The expression $m \% n$ yields the integer remainder after $m$ is divided by $n$.
- Modulus is an integer operation -- both operands MUST be integers.
- Examples: $17 \% 5 = 2$
  - $6 \% 3 = 0$
  - $9 \% 2 = 1$
  - $5 \% 8 = 5$

Detailed Modulus Example

- $17 \% 5 = 2$

The whole number left over (remainder) is the answer.
Another Detailed Modulus Example

- $5 \% 8 = 5$

The whole number left over (remainder) is the answer.

$8 \overline{5}$

$\underline{0}$

$8 | 5$

$- 0$

$R(5)$

Uses for Modulus

- Used to determine if an integer value is even or odd
  
  $5 \% 2 = 1$ odd  
  $4 \% 2 = 0$ even

  If you take the modulus by 2 of an integer, a result of 1 means the number is odd and a result of 0 means the number is even.

- The Euclid's GCD Algorithm (from the Algorithms 1 lecture)

Arithmetic Operators

Rules of Operator Precedence

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Precedence &amp; Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Evaluated first. If nested (embedded), innermost first. If on same level, left to right.</td>
</tr>
<tr>
<td>* / %</td>
<td>Evaluated second. If there are several, evaluated left to right.</td>
</tr>
<tr>
<td>+ -</td>
<td>Evaluated third. If there are several, evaluated left to right.</td>
</tr>
<tr>
<td>=</td>
<td>Evaluated last, right to left.</td>
</tr>
</tbody>
</table>

Using Parentheses

- Use parentheses to change the order in which an expression is evaluated.

  $a + b \ast c$  
  
  Would multiply $b \ast c$ first, then add $a$ to the result.

  If you really want the sum of $a$ and $b$ to be multiplied by $c$, use parentheses to force the evaluation to be done in the order you want.

  $(a + b) \ast c$

- Also use parentheses to clarify a complex expression.