Introduction To Full-Stack Web Development

CS 386

Michael Kremer

Last updated: 9/27/2023 7:04:19 AM

Class 10

- 10.1 Variables
- 10.2 Operators
- 10.3 If statement
- 10.4 Loops

10.1 Variables

- Declarations
  - JavaScript has three kinds of variable declarations:
    - **var**: Declares variable, optionally initializing it to value (older way, hoisted to top)
    - **let**: Declares block-scoped, local variable, optionally initializing it to value (modern declaration)
    - **const**: Declares block-scoped, read-only named constant, must initialize
  - **Important:**
    - Do not use var declaration anymore!
    - Still see var in lots of code including libraries

10.1 Variables

- Variable scope
  - Variable may belong to one of following scopes:
    - Global scope: Default scope for all code running in script mode
    - Module scope: Scope for code running in module mode (using type="module" in browser)
    - Function scope: Scope created within function
  - In addition, variables declared with let or const can belong to additional scope:
    - Block scope: Scope created with pair of curly braces (block)
  - When declaring variables outside of any function → called global variable
    - Available to any other code in current document
  - When declaring variables within function → called local variable
    - Available only within that function
  - Always use smallest scope needed:
    - Simplifies program
    - Enhances performance
    - Easier to debug

10.1 Variables

- **Example 10-1:**
  - Create constant PI assigning pi from Math library
  - Declare variable var_global using let, no assignment
  - Display var_global in alert (notice value of undefined)
  - Create block scope using curly braces:
    - Within declare variable var_block using let, assign it number value, display in alert
  - After block scope, declare variable radius using let assigning it value of 2
  - Display area of circle in alert using PI and variable radius
  - Display variable var_block in alert → generates error message
  - Assign new value to constant PI → generates error message

10.1 Variables

➢ Example 10-1:

C:\Temp\CS386\Example10-1.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example10-1.js

```
1  const PI = Math.PI; //Global scope constant
2  let var_global; //Global scope variable
3  alert('Variable var_global = ' + var_global);
4  //Block scoped variable
5  {
6      let var_block = 5;
7      alert('Variable var_block =' + var_block);
8  }
9  //Area of circle
10 let radius = 2;
11 alert("Area of circle = " + PI * radius ** 2);
12
13 alert('Variable block =' + var_block);//Out of scope --> error
14 PI = 5; //Cannot reassign new value to constant
```

length : 436 lines : 19 Ln : 19 Col : 1 Pos : 437 Windows (CR LF) UTF-8 INS

about:newtab

Variable var_global = undefined

OK

about:newtab

Variable var_block =5

Don't allow about: to prompt you again

OK

about:newtab

Area of circle = 12.566370614359172

Don't allow about: to prompt you again

OK

Uncaught ReferenceError: var_block is not defined
<anonymous> debugger eval code:13
[Learn More]

Uncaught TypeError: invalid assignment to const 'PI'
<anonymous> debugger eval code:14
[Learn More]

10.2 Operators

- Already introduced arithmetic operators
- These are binary operators → act on two operands
- There are also unary arithmetic operators:
- Unary plus (+)
  - Converts its operand to number (or to NaN) and returns that converted value
  - When used with operand that is already of number type, it does not do anything
- Unary minus (-)
  - Converts its operand to number type, if necessary, and then changes sign of result

10.2 Operators

- Increment (++)
  - Increments (i.e., adds 1) to its single operand:
    - Converts its operand to number type
    - Adds 1 to that number
    - Assigns incremented value back into variable, element, or property
- Decrement (--)
  - Same as increment, except it subtracts:
    - Converts value of operand to number type
    - Subtracts 1 from that number
    - Assigns decremented value back to into variable, element, or property

10.2 Operators

- Return value of the ++/-- operator depends on its position relative to operand:
  - When used before operand (**pre**-increment/decrement operator):
    - Increments/decrements operand
    - Evaluates to incremented/decremented value of that operand
  - When used after operand (**post**-increment/decrement operator):
    - Increments/decrements its operand
    - BUT evaluates to unincremented/undecremented value of that operand
- <u>Example:</u>
  - let i = 1, j = ++i;
    - i and j are both 2
  - let i = 1, j = i++;
    - i is 2
    - j is 1

10.2 Operators

- Relational Operators
  - == and === operators check whether two values are same
  - Using two different definitions of "sameness":
    - Both operators accept operands of any type
    - Both return true if their operands are equal and false if they are different
    - === operator is known as strict equality operator (or sometimes identity operator):
      - Checks whether its two operands are "identical" using stricter definition of sameness
    - == operator is known as equality operator:
      - Checks whether its two operands are "equal" using more relaxed definition of sameness that allows type conversions

10.2 Operators

- Relational Operators
  - != and !== operators test for exact opposite of == and === operators:
    - != inequality operator:
      - Returns false if two values are equal to each other according to ==
      - Returns true otherwise
    - !== strict inequality operator:
      - Returns false if two values are strictly equal to each other
      - Returns true otherwise
  - <u>Example:</u>
    - “1” == true //evaluates to true
    - “1” === true //evaluates to false

10.2 Operators

- Relational Operators
  - Operands of relational operators may be of any type
  - Comparison can be performed only on numbers and strings (dates are numbers!)
  - Operands that are not numbers or strings are converted

Relational Operator	Description
Less than (<)	The < operator evaluates to true if its first operand is less than its second operand; otherwise it evaluates to false.
Greater than (>)	The > operator evaluates to true if its first operand is greater than its second operand; otherwise it evaluates to false.
Less than or equal (<=)	The <= operator evaluates to true if its first operand is less than or equal to its second operand; otherwise it evaluates to false.
Greater than or equal (>=)	The >= operator evaluates to true if its first operand is greater than or equal to its second operand; otherwise it evaluates to false.

10.2 Operators

- Logical Operators
  - logical operators &&, ||, and ! perform Boolean algebra
  - Often used in conjunction with relational operators to combine two relational expressions into one more complex expression
  - JavaScript performs short-circuiting of multiple conditions:
    - For Boolean AND (&&):
      - If first condition is false, it does not evaluate any further conditions
      - When one condition is false, resultant condition will be false
    - For Boolean OR (||):
      - If first condition is true, it does not evaluate any further conditions
      - When one condition is true, resultant condition will be false
  - <u>Example:</u>
    - a is undefined variable → error
    - In short-circuit expression error is not raised!

```
>> a < 5;
Uncaught ReferenceError: a is not defined
    <anonymous> debugger eval code:1
  [Learn More]
>> 5 < 2 && a < 5;
<- false
```

10.2 Operators

➢ Example 10-2:

- First condition is false:
  - Stops evaluating second condition
  - a is not incremented

C:\Temp\CS386\Example10-2.js - Notepad++

Example10-2.js

```
let a = 1; //Initialize variable a
//First condition is false, stops evaluating further conditions
let b = (5 < 3 && a++); //Assigns false, a is not incremented
alert('a = ' + a + "  b = " + b);
```

length : 207 lines : 9 Ln : 9 Col : 1 Pos : 208 Windows (CR LF) UTF-8 INS

about:newtab

a = 1 b = false

OK

- First condition is now true:
  - continues evaluating second condition
  - a is incremented

C:\Temp\CS386\Example10-2a.js - Notepad++

Example10-2a.js

```
let a = 1; //Initialize variable a
//First condition is now true, continues evaluating further conditions
let b = (5 > 3 && a++); //Assigns a, a is incremented
alert('a = ' + a + "  b = " + b);
```

length : 206 lines : 9 Ln : 9 Col : 1 Pos : 207 Windows (CR LF) UTF-8 INS

about:newtab

a = 2 b = 1

OK

10.2 Operators

- Return value of Boolean expressions
- && and || operators will return value of specified operand:
  - If operand is using non-Booleans value, returns non-Boolean value
  - Otherwise, returns true or false
- AND "&&" returns first falsy value
- OR "||" returns first truthy value
- <u>Examples:</u>
- Remember:
  - 5 = truthy value
  - 0, null = falsy values

```
console.log(`5 && 0 && null --> ${5 && 0 && null}`);
console.log(`0 && 5 && null --> ${0 && 5 && null}`);
console.log(`5 || 0 || null --> ${5 || 0 || null}`);
console.log(`0 || 5 || null --> ${0 || 5 || null}`);
```

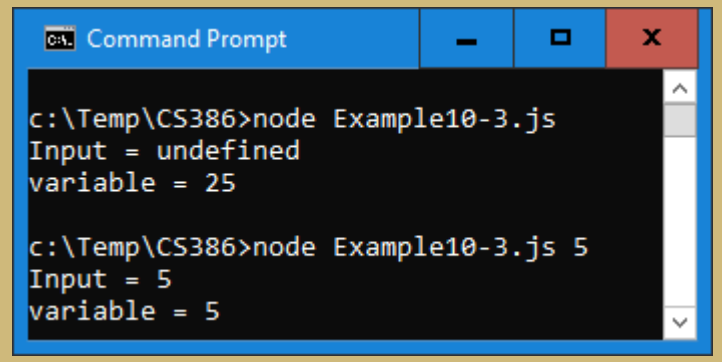
```
5 && 0 && null --> 0
0 && 5 && null --> 0
5 || 0 || null --> 5
0 || 5 || null --> 5
```

10.2 Operators

- **Example 10-3:**
- Declare variable input
- Use try..catch to solicit number input (prompt in browser, argument vector in node)
- After try..catch block display variable input in console
- Declare constant INITIAL and assign value 25
- Declare variable named variable:
  - Assign either input (if truthy)
  - Otherwise, use INITIAL
- Display value of variable in console:

```
Input = 0
variable = 25
```

```
Input = 32
variable = 32
```



10.2 Operators

- **Example 10-3:**

C:\Temp\CS386\Example10-3.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example10-3.js

```
let input; //Declare input variable
try { //Browser input using prompt
    input= +prompt('Enter a number');//No default specified
} catch { //node.js input using argument vector
    input = process.argv[2];
}
console.log('Input = ' + input); //Display input
const INITIAL = 25; //Declare and initialize constant INITIAL
let variable = input || INITIAL; //Return first truthy value
console.log('variable = ' + variable); //Display variable
```

length : 453 lines : 15 | Ln : 14 Col : 1 Pos : 452 | Windows (CR LF) | UTF-8 | INS

10.2 Operators

- Assignment Operator
  - To assign a value/expression into variable, use single equal operator
  - This operator has right-to-left associativity:
    - Evaluate right-side first, then assign result into left side
  - Also means left side variable can be on right side to accumulate current value
  - Can be simplified by using assignment with operation as one operator (+=)

Syntax:
var_name **=** *val/exp*

Syntax:
var_name **=** *var_name* **+** *val/exp*

Syntax:
var_name **+=** *val/exp*

Operator	Example	Equivalent
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

10.3 if statement

- If statements
  - Use if statement to execute statements if logical condition is true
  - Use optional else clause to execute statement if condition is false
  - The following values evaluate to false (also known as Falsy values):
    - false
    - undefined
    - null
    - 0
    - NaN
    - Empty string ("")
  - All other values—including all objects—evaluate to true when passed to conditional statement

Syntax:

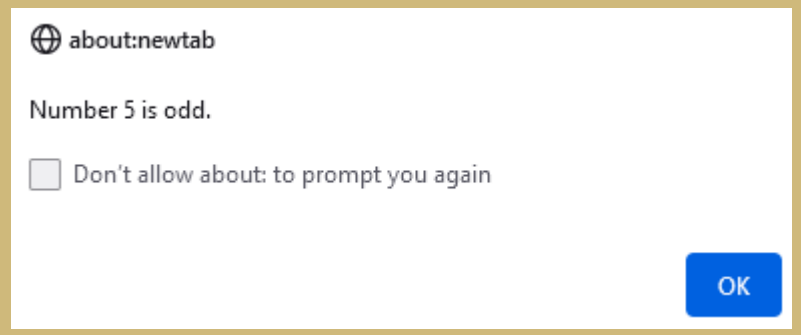
```
if (condition) {
  statement1;
} else {
  statement2;
}
```

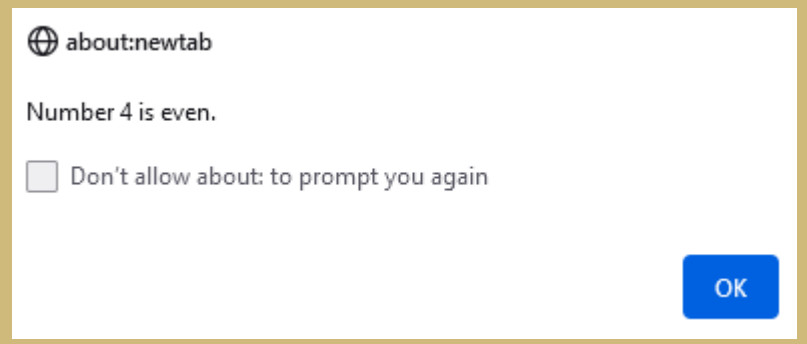
Syntax:

```
if (condition1) {
  statement1;
} else if (condition2) {
  statement2;
} else {
  statement3;
}
```

10.3 if statement

- **<u>Example 10-4:</u>**
- Create variable number and assign user inputted number using prompt
- Use modulo operator (%) to determine whether number is even or odd
- Produce alert outputs as shown below





10.3 if statement

- **Example 10-4:**

C:\Temp\CS386\Example10-4.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example10-4.js

```javascript
let number = prompt("Enter whole number: ", 0); //Get number from user
//No number conversion performed on prompt!!
//Determine whether number is even or odd
if (number % 2 === 0) { //No remainder, implicit conversion of number
    alert("Number " + number + " is even.");
} else { //remainder
    alert("Number " + number + " is odd.");
}
```

JavaScrip length : 350 lines : 13 | Ln : 10 Col : 1 Pos : 345 | Windows (CR LF) | UTF-8 | INS

10.4 Loops

- Loop Statements (for loop)
  - for loop repeats until a specified condition evaluates to false
  - JavaScript for loop is similar to Java and C for loop

Syntax:

for ([*initialExpression*]**;** [*conditionExpression*]**;** [*incrementExpression*]**) {**
  statement(s)
}

  - initialExpression:
    - Initializing expression, if any, is executed (only once at very beginning)
    - Expression usually initializes one or more loop counters, but syntax allows expression of any degree of complexity
    - Expression can also declare variables
  - conditionExpression:
    - If value of conditionExpression is true, the loop statements execute
    - Otherwise, for loop terminates
    - If conditionExpression expression is omitted entirely, the condition is assumed to be true
    - Then statement(s) executes
  - incrementExpression
    - If present, update expression incrementExpression is executed

10.4 Loops

- **Example 10-5:**
  - Create for loop using step as increment variable
  - Loop 5 times, incrementing by one
  - Produce the following output in the console

```
Walking step 0
Walking step 1
Walking step 2
Walking step 3
Walking step 4
```

10.4 Loops

- **Example 10-5:**

C:\Temp\CS386\Example10-5.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example10-5.js

```
for (let step = 0; step < 5; step++) {
    // Runs 5 times, with values of step 0 through 4
    console.log('Walking step ' + step);
}
```

length : 143 lines : 10 Ln : 7 Col : 1 Pos : 138 Windows (CR LF) UTF-8 INS

10.4 Loops

- Loop Statements (while loop)
  - while statement executes its statements as long as specified condition evaluates to true
  - Condition is checked before each iteration

Syntax:

```
while (conditionExpression) {
    statement(s)
}
```

  - Variation of while loop: do … while loop
  - Place condition at bottom of loop
  - Condition is checked after each iteration
  - Ensures that loop is executed at least once regardless of condition
  - **IMPORTANT:** Must end in semicolon

Syntax:

```
do {
    statement(s)
} while (conditionExpression) ;
```

10.4 Loops

- **Example 10-6:**
  - Create two variables, n and x initialized to 0
  - Create while loop using condition of n less than 3
  - Inside loop:
    - Increment n by one
    - Accumulate x by n
    - Output n and x in console

```
n = 1   / x = 1
n = 2   / x = 3
n = 3   / x = 6
```

10.4 Loops

- **Example 10-6:**

C:\Temp\CS386\Example10-6.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? + ▼ ×

Example10-6.js

```
let n = 0; //Initialize n
let x = 0; //Initialize x
while (n < 3) { //Loop as long as n is less than 3
    n++; //Increment n by one
    x += n; //Accumulate x by n
    console.log("n = " + n + "    / x = " + x);
}
```

length : 220 lines : 12 | Ln : 10 Col : 1 Pos : 217 | Windows (CR LF) | UTF-8 | INS