Week 3

61

Week 2 Review

A CSS style rule consists of _____, _____ and _____

An internal stylesheet uses the _____ tag.

The CSS selector: UL,LI will apply to _____
What if it is changed to: UL LI _____

position: absolute places an element relative to _____

JavaScript code is executed at the _____

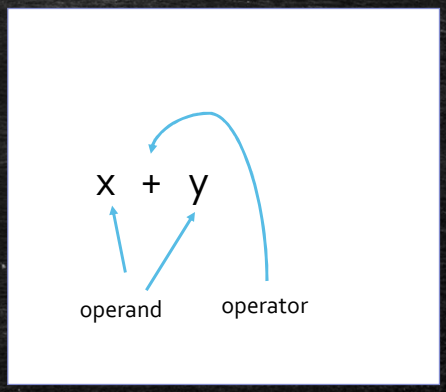
I want to create a variable called "n" within an if statement that will only be accessible within the if statement – how can I make this happen?

In the expression: `exp1 ? result1 : result2;`
exp1 is a/an _____

62

Operators

- Arithmetic
- Arithmetic with assignment
- Assignment
- Comparison
- Logical
- Concatenation



63

63

Arithmetic

+	addition	Adds numeric operands.
-	subtraction	Used for negating or subtracting.
++	increment	Add 1 to the operand.
--	decrement	Subtract 1 from the operand.
*	multiplication	Multiplies two numerical operands.
/	division	Divides first operand by second operand
%	modulus	Calculates the remainder of first operand divided by second operand.

64

64

Operator Precedence

- JS follows PEMDAS
- "Please Excuse My Dear Aunt Sally"
- Parentheses
- Exponent
- Multiplication, Division
- Addition, Subtraction
- See: https://www.dummies.com/article/technology/programming-web-design/general-programming-web-design/javascript-operator-precedence-254119

65

65

Example: Tip Calculator

```
var checkAmount, tipAmount;
const TIP_PERCENT=.17;
checkAmount = prompt("What is the check amount? ","20");
document.write("Check amount: $"
                + checkAmount
                + "Tip: $ "
                + checkAmount * TIP_PERCENT);

// why isn't parseInt needed?
```

66

66

Example: Is a number even?

```
var num, result;
num = prompt("Enter a number", "5");
result = num % 2;
document.write ("The remainder of " + num + “divided by 2 is " + result);
```

67

67

Compound Assignment Operators

- +=, -=, *=, /=, %=
- Performs the operation and does the assignment
- Example:
  `x = x + 3;`
  is the same as
  `x += 3;`

Four ways to add 1 to a value

```
x = x+ 1;
x++;
++x;
x += 1;
```

68

68

Comparison

== equality	True if two operands have the same value (data conversion may occur)
!= inequality	Opposite of the equality (==) operator.
> greater than	True if first operand is larger
>= greater or equal	True if first operand is greater than or equal to a second operand
< less than	True if first operand is smaller
<= less than or equal	True if first operand is less than or equal to a second operand

69

69

Conditional Operator ?:

- test ? result1 : result2
- Do the test, if the result is true, then the value of the operator is Result1 otherwise it is Result2
- Remember: this is an operator, there is no implied assignment
- "Absolute value" example:
  - `n = n<0 ? -n : n;`

70

70

Example: Even Revisited

```
var num, result;
num = prompt("Enter a number", "5");
result = num % 2;
document.write ("The number: " + num
           +  (result==0 ? " is " : " is not ")
           + "even";
```

71

71

Logic Operations

- Logical operands "glue" two comparison operators together.
- JavaScript supports three logical operations:
  - && AND (shift-7) true when two operands are **both** true
  - || OR (shift \) true when **either** of two operands is true
  - ! NOT (shift 1) (opposite) true when an operand is false
- For example, there is no "between" operator, but you can accomplish the same thing with AND

```
between = z>10 && z<=20;   // true when z is 11 through 20 inclusive
```

72

72

Logic Operations (continued)

- When a value of one input expression forces the output result of a logical operator, it is called a "Forcing Function"
  - The forcing function for AND is false.
  - The forcing function for OR is true.
- Shortcut calculation- if there are two parts to an expression and the first part forces the value, the second part is not evaluated

```
x = (6<3 )&& (7<4);  // no need to evaluate (7<4)
```

73

73

Mixing Numbers, Strings and Operators

- + operator
  - string + string — concatenation
  - string + number — convert number to string and then concatenate
  - number + number — addition
- Comparison operators (ex, == >)
  - string > string — alphabetical order
  - string > number — convert string to number and compare numerically
  - number > number — numerical order

74

74

Figure it Out

- What is:
  `3 > 4 && 6 < 10`

  `“6” == 6 || 8< 0`

  `! (5 > 2)`

- Assume n = 10 for each what is:

  `n > 0`

  `n <= 5 && n >= 10`

  `n >= 1 && n<= 10`

  `n == “10” || n < 3`

75

75

Prefix vs Postfix Notation

- x++
  value is x
  add one to x

- ++x
  value is x+1
  add one to x

Given, x = 3

```
y = x++;     //y = 3,  x = 4
y = ++x;     //y = 5,  x = 5
```

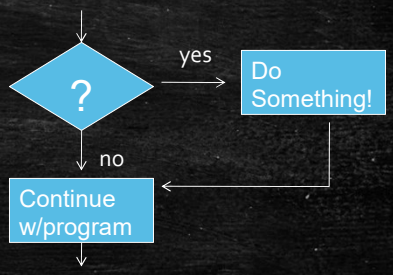
76

76

Conditionals

- The IF construct helps determine PROGRAM FLOW based on a yes/no question.

- `if (n>1)`
  `alert ("Do something");`



77

77

Simple IF statement

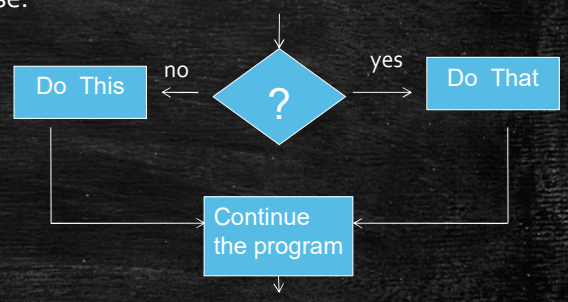
- `if (n>=0)`
  `alert("N is a positive number");`
- Following the keyword "if" is a *conditional expression* in parenthesis - an expression that is evaluated as true of false
  - Comparison operators
  - Boolean logic operators (and, or)
  - Anything can be evaluated as true/false (recall that zero is false)
- If the expression is true, then do the statement immediately following the "if"
  Otherwise, skip that statement.
  - Several statements can be included in an "if" block by enclosing them in a code block { . . . }

78

78

Otherwise …

- Sometimes we want to take different actions when the conditional expression is true vs when it is false.
  - Use an ELSE statement (think "otherwise"_
- ```
  if (n>=0)
      alert("n is a positive number");
  else
      alert("n is a negative number");
  ```
- You can do several statements after the else by enclosing them in a code block { . . . }



79

79

Notes

- Each part of an if / else if/ else statement is mutually exclusive. i.e., only one can be true.
- *Do not* put a conditional expression after a lone "else"
  ie: else (n>3) //WRONG!
- You must use curly brackets when there is more than one statement after an if or else. Curly brackets are optional for a single statement.

80

80
  ```

*Which code has a **syntax ERROR:***

```
a. if (a<b)
   document.write( "hello" );

b. if a<b
   document.write( "hello" );

c. if (a<b) && (b<c)
  {
    document.write( "hello" );
    document.write( "there" );
  }

d. if (a<b) && (b<c)
   document.write( "hello" );
   document.write( "there" );

e. if (a=4)
     document.write( "hello" );
```

81

81

Examples: Solve using if/else statements

- If a guess matches a number, display "You are correct", otherwise display "Sorry, try again"

```
if (guess==number)
        alert("correct");
else
        alert("Sorry, try again");
```

- If a guess matches a number, display "You are correct", otherwise display "Try again – you may have three guesses" and add one to a variable called numGuesses.

```
if (guess==number)
        alert("correct");
else
{
        alert("Try again – you may have three guesses");
        numGuesses+= 1;
}
```

82

82

Switch – shorthand for IF

- ```
  switch(expression)
  {
     case value1: do this; break;
     case value2: do that; break;
     default: do the other thing; break;
  }
  ```
- *break* keeps it from "falling through"
- *default* placed at the end and is optional

83

83

Problem: solve the following using switch

- You have a variable called coin and a variable called value.
- If coin is "nickel", value is 5
- If coin is "dime", value is 10
- If coin is "quarter" value is 25
- Create a switch statement that determines the value of the coin

```
switch(coin)
{
   case "nickel" :
      val = 5;
      break;
   case "dime" :
      val = 10;
      break;
   case "quarter" :
      val = 25;
}
```

84

84
```

Figure it out

- What is the value of x at the end?

```
y = 5;  x= 7;
switch(y)
{
  case 1:   x= x*2; break;
  case 5:   x -= y;
  case 10:  x++; break;
  default:  x = 17;
}
```

85

85

Repetition using Loops

- **Two types of loops: Counting and Waiting**
  - In a **counting loop**, you do something for a specified number of times.
    - For example – display "Hello World" on the screen 10 times.
    - Or maybe, move two squares – 4 times.
    - *In JavaScript, a **for** loop is optimized for counting*
  - In a **waiting loop**, you do something until something happens.
    - For example – get numbers from the user until the user enters: -1
    - Or, move a robot forward 1 inch at a time–until a boundary is detected.
    - *In JavaScript, a **while** loop is optimized for waiting.*

86

86

FOR Loop Syntax

```
for (init ; test ; update)
{
  // loop statement(s)
}
```

Display the numbers: 1 to 10

```
for (n=1; n<=10; n++)
  alert(n);
```

Notes:

- **initialization**- statement that occurs prior to any iteration
- **test** - conditional expression that is evaluated at the beginning of each iteration. When expression evaluates to false, exit the loop
- **update** - statement that occurs at the end of each iteration. Often used to update a counter.
- *Any or all of these can be omitted*
  - for (;;) is an intentional infinite loop
- Do not put a semicolon at the end of a "for" header

87

87

WHILE loop syntax

- Continue to *iterate* as long as the conditional expression is true
- Alternate form: do-while allows for at least one iteration through the loop

```
while (test)
{
  // loop statement(s)
}

do  {
  // loop statement(s)
} while (test);
```

88

88

Example: display the numbers from 1 to 10

```
n = 1;              // initialization
while (n<=10)       // test
{
  alert(n);
  n++;              // update
}
```

Notes:

- Do not put a semicolon at the end of a while header *except* in a do-while construct.
- while(true) is an intentional infinite loop

89

89

Figure it Out

How many times will "hello" be printed?

```
for (i=1; i<=5; i++)
  document.write( "hello" );
while (i<= 5)
  document.write( "hello" );
```

How many times will "hello" be printed?

```
for (i=2; i<=5; i++);
  document.write( "hello" );
```

90

90

Break And Continue

- Break and continue statements provide additional control for directing the flow through a loop.
- **Break-** exits a loop.
  - Generally used as part of an *if* construct.
  - Used to provide an alternate place to exit from the loop
  - Use carefully to avoid unreadable code.
- **Continue-** ends the current iteration of a loop.
  - In a while loop, continue goes directly to the *test*
  - In a for loop, continue goes directly to the *update*.

91

91

Break and Continue

```
for (i=1; i<=5; i++)
{
  if (i==2) continue;
  document.write( "hello"
  );
  if (i==4) break;
}
```

```
i=0;
while (i<=5)
{
  if (i==2) continue;
  document.write( "hello"
  );
  if (i==4) break;
}
```

92

Example

```
//this loop demonstrates two exit points
  while(true)
  {
       if (x == 10)  break;          //get out here
       if (x == 20)  break;          //or get out here
       x++;
  }
// special case for first iteration
  for (x=0; x<10; x++)
  {
             document.write(x);
             if (x==0)
                    continue;
       // more loop statements can go here
  }
```

93

93

Try It

- Solve the following with a for loop and then with a while loop:
  - Count down by two's from 40 to 10. i.e., 40,38,36,34,… (display each number).
- Solve the following using a while loop
  - Use prompt to get numbers from the user until the sum of the numbers exceeds 20

94

Functions

- A function is a named set of tasks that are not executed until the function is *called*
- Functions can be *anonymous* (no name) and are used as a parameter to another function.
- Functions can have inputs, outputs, and byproducts
- Use the *return* statement to end the function or return a value

```
function add1(p1,p2)
{
  var a = p1, b = p2, sum;
  sum = a + b;
  alert ("The sum is: " + sum);
}

add(3, 4);
```

95

95

The return statement

```
function add2()
{
  var a = 2, b = 3, sum;
  sum = a + b;
  return sum;
}
```

Call the function

```
var sum;
sum = add2();
alert("The sum is: " + sum);
```

96

96

Providing Arguments (Parameters)

```
function add3(a,b)
{
  var sum = a + b;
  alert(sum)
}
```

Calling the function

```
add3(2,3);
```

97

97

Providing arguments and returning a value

```
function add4(a,b)
{
  var sum = a + b;
  return sum;
}
```

Calling the function

```
var sum;
sum = add4(2,3);
alert ("The sum is: " + sum);
```

98

98

Figure it out

```
function process( x, z)
{
   x+= 10;
   z =z + x;
   return z;
}
```

```
//What is displayed?

x = 20;
a = 10;  b= 11;
a= process(a, b);
alert(x);
alert(a);
```

99

99

Default Values

- Default values can be provided for arguments whose value is not provided
- function hasDefaults (a, b=10, c=5)
  {
  console.log("A: " + a + "B: " + b + "C: " + c);
  }

  //what is displayed?
  hasDefaults(7,3);

100

Events and Event Handlers

- Using the JavaScript event property
  - `btn1.onclick = "doSomething";`
  - `btn1.onclick = function()  // anonymous function`
    `{ /* do something here */ }`
- Add an event listener
  - `btn1.addEventListener("click", doSomething);`
  - More than one event handler can be associated with an event.
- Accessing an object to associate with the event handler
  - document.getElementById("the-id"); //object with that id
  - document.querySelector('.the-class'); //first object with that class
  - document.querySelectorAll('.the-class'); //collection of objects with that class

101

101

Event Handlers

- Assign event handlers when all elements have been loaded

```
window.onload= function()
 {
                    // do event associations here
 }
```

The keyword, *this*, can be used to *refer* to the object that invoked the event

102

102

Example

HTML

```
<div class="button1" id="button1-id" name="button1-name" >Press Me</div>
<div id="result"></div>
```

CSS

```
body,html {font-size: 30px;}
#button1-id {border:2px solid #000; padding: 15px; width: 100px;  text-align: center; }
#result {font-size: 19px; margin-top: 20px;}
```

JS

```
window.onload= function() {
        btn = document.getElementById("button1-id");
        btn.addEventListener("click",function(){
                result = document.getElementById("result");
                result.innerHTML = "Content: " + this.innerText
                + "<br />Class " + this.className
                + "<br />Name " +this.getAttribute("name")
                + "<br />ID " +  this.id;
        })  // end button click
} // end window onload
```

103

String object

- length — number of characters in a string
- charAt() — returns the character at the specified index
- concat() — joins two or more strings, and returns a copy of the joined strings
- indexOf() — returns the position of the first occurrence of a specified string
- lastIndexOf() — returns the position of the last occurrence of a string
- slice() — extracts a part of a string and returns a new string
- split() — splits a string into an array of substrings
- substr() — gets a substring defined by a start position and a number of characters
- substring() — gets a substring defined by a start and end index
- toLowerCase() — returns the string in lower case
- toUpperCase() — returns the string in uppercase

104

104

Example revisited

```
HTML
<div class="theButton btn1" >Press Me</div>
<div class="theButton btn2" >Press Me</div>
<div class="theButton btn3" >Press Me</div>
<div id= "result"></div>

JS
window.onload= function() {
        for (i=1; i<=3; i++)
          document.querySelector(".btn" + i).addEventListener("click",showNumber);
} // end window onload

function showNumber()
{
        result = document.getElementById("result");
        theClass = this.className;
        index = theClass.indexOf("btn")+3;
        number = parseInt(theClass.substring(index));
        result.innerHTML = this.className + "<br />" + number * 2;
}
```

105

Arrow Functions

- Use the arrow as a shortcut to define and then call a function
- You can have a function serve as a parameter – similar to the concept of a function pointer
- Assume you want a simple function as follows:
  function hello() { return "Hey there!"; }
  - Call the function using: hello()
- Simplify to:
  hello = () => {return "Hello World!"};
- Simplfy further to:
  hello = () => "Hello World!";
  - Call the function using: hello()
- Add parameters:
  hello = (num) => "Two times " + num + " is " + 2 * num;

106

Example

```
add = (a,b) => a+b;

sub = (a,b) => a-b;

function operate (a,b,op)

{
        return op(a,b);
}

operate (4,7,add);

//Could you implement operate using an arrow function?
```

107

Arrays in javascript

- Arrays are implemented with the Array object.
- An array is a group or collection of items that are referenced using a group name and an index
- The first index is 0
- Array elements can be differing types
- The size of the array is *dynamic*
- Create an Array:
  - Use the *Array* object
    `things = new Array(1,2,3);`
  - Use literal notation [...]
    `things = [1,2,3];`

108

108

A conceptual view …

- numbers = [1,2,3,4,5];

1
2
3
4
5

- *Create an empty array with 5 elements*
  - var numbers = new Array(5);

- document.write (numbers[2]);
  - Displays the 3rd element in the array – in this case "3"

109

109

Common properties and methods of Array object

- Helpful properties:
  - length
- Helpful methods:
  - indexOf
  - join
  - push/pop
  - forEach
  - sort
  - map

110

110

USING the Array object

- Create an empty array — var arr = []; // arr is empty
- Add elements to the end — arr.push(1,2,3); // arr = 1 2 3
- Change the 3rd element — arr[2]=5; // arr = 1 2 5
- Add an element in the last position — arr[3]=4; // arr = 1 2 5 4
- Sort an array — arr = arr.sort(); // arr = 1 2 4 5
- Join to a string — alert(arr.join(' * ');
  //displayed: 1 * 2 * 4 * 5 *

111

111

Loops and Arrays

- A loop counter can be used to iterate through an array.
  - To display the array
  - To do something to every element
  - To give a value to every element

```
//add 3 to each item
numbers = [2,4,6,8,10];
for (i=0; i<numbers.length;i++)
   numbers[i] += 3;
```

112

112