1 **Arrays**

JavaScript

2 **An Array (Page 1)**

- A collection of data elements storing values which share the *same name*
- They are stored in *contiguous* (consecutive) locations in RAM
- Each piece of data contained in an array is called an element
- Element values in the array may be of *different data types*

3 **An Array (Page 2)**

- Individual elements of the array are uniquely identified by an index (or subscript)
- Format:

  arrayName[index]
  - The index is an integer (or integer variable) in *brackets* representing element's position in array
- Examples:

  cars[0] cars[3] cars[index]
  - The range of values of an index is from zero (0) to one less than the size of the array

4 **The Subscript Brackets**

- The brackets surrounding index are JavaScript operators, e.g.

  primes[0]
- Have the highest precedence of all operators
- Example:

```
index = 9;
x = (a + b) / primes[index – 2];
```

5 **Declaring an Array (Page 1)**

- The array is an *object* and like any variable, it must be declared
- When declared, default values are:
  - For primitive type elements, *numerics* are zero (0) and booleans are false
  - For references (non-primitives), values are null

6 **Declaring an Array (Page 2)**

- Most common (and preferred/easiest) way to declare an array is to assign a comma-delimited array literal in [brackets] to the array
- Format:

  var *arrayName* = [*value1*, *value2*, ...];
- Example2:

```
var primes = [2, 3, 5, 7, 11];
var cars = ["Ford", "Chevy", "Honda"];
```

7 **Declaring an Array (Page 3)**

- An array object also can be instantiated using operator new and the Array() constructor
- Formats:
  var *arrayName* = new Array();
  - Creates an empty array with no elements
- The same as:
  var *arrayName* = [];

8 **Declaring an Array (Page 4)**

- Example:
  var a = new Array();
  - Creates empty an array named 'a' with no elements
- The same as:
  var a = [];

9 **Declaring an Array (Page 5)**

- Additional formats using Array() constructor:
  var *arrayName* = new Array(*int*);
  - Creates an array with the number of elements as specified by the *int* argument
  var *arrayName* = new Array(*value1*, *value2*, ...);
  - Creates an array which specifies the explicit values for two or more elements

10 **Declaring an Array (Page 6)**

- Examples:
  var primes = new Array(5);
  - Creates an array named 'primes' with five elements
  var cars = new Array("Ford", "Chevy", "Honda");
  - Creates an array which specifies the explicit values for two or more elements

12 **Commas in the Array Literal**

- If a value is omitted in an array literal, the omitted literal is given the value undefined:
  var values = [2, , 6];
- Syntax of the array literal allows for a *trailing* comma
  var values = [2, , 6,];
  - Creates just three elements

14 **The length Property of an Array**

- The Array.length property stores an integer which represents the *number of elements* in the array
- Format:
  array.length

- Examples:
  for (index = 0; index < cars.length; index++)
  var primes = justPrimes.length
  [2, 3, 5, 7, 11].length // returns 5

15 **Updating the length Property**

- The length property can be changed
  - Setting the length value to less than that of the current length of the array reduces the size of the array (removes the right-most elements)
- Format:
  array.length = *integerLiteral*;
- Example:
  primes = [2, 3, 5, 7, 11];
  primes.length = 3; // primes is now [2 ,3, 5]
  primes.length = 7; // primes is now [2 ,3, 5, ,]

16 **Iterating through an Array**

- The most common way to loop through the elements of an array is with a for loop

```
var index;

for (index = 0; index < myArray.length; index++)
{
   document.write( myArray[index] );
}
```

17 **The for...in Loop for Arrays**

- An alternate method for iterating through an array is the for...in loop
  - Iterates through (increments) the indexes of the array

```
var index;

for (index in myArray)
{
   document.write( myArray[index] );
}
```

19 **The push() method**

- The Array.push() method *inserts* one or more new items to the *end* of an array
- Format:
  string.push(*item1*, *item2*, ...);
- Example:
  var cars = ["Ford", "Chevy", "Honda", "Audi"];
  cars.push("Mercedes", "Toyota");

20 **The delete Operator (Page 1)**

- The delete operator removes any element from the array
- Using delete on array elements leaves undefined *holes* in the array (leaves element undefined)
  - It may be better to use pop() and/or shift() instead

21 **The delete Operator (Page 2)**

- Format:
  delete *arrayName*[*index*];
- Example:
  delete cars[1];

23 **The reverse() Method**

- The Array.reverse() method reverses the order of the elements in the array
  - The first becomes the last and the last becomes the first, etc.
- Format:
  arrayName.reverse();
- Example:
  cars.reverse();

24 **The sort() Method**

- The Array.sort() method sorts the array numerically or alphabetically
- Format:
  arrayName.sort();
- Example:
  cars.sort();

26 **The concat() Method**

- The Array.concat() method joins two or more arrays and *returns* a copy of the joined arrays
- Format:
  arrayName1.concat(*arrayName2*);
- Example:
  var cars = ["Ford", "Chevy", "Honda", "Audi"];
  var primes = [2, 3, 5, 7, 11];
  cars = cars.concat(primes);

27 **The join() Method (Page 1)**

- The Array.join() method joins all elements of the array into a single string
- By default the list is comma-delimited with no spaces between the characters
  - Or the programmer can define the delimiter

28 **The join() Method (Page 2)**

- Format:
  arrayName1.join([*string*]);
  - The optional *string* argument will be inserted between each element of the joined string
- Examples:
  document.write(cars.join());
  - Displays the string "Ford,Chevy,Honda,Audi"

  document.write(cars.join(", "));
  - Displays the string "Ford, Chevy, Honda, Audi"

30 **The slice() Method (Page 1)**

- The Array.slice() method selects part of an array and returns the new array
- The method leaves the original array unchanged
- A negative number counts from the back of the array

31 **The slice() Method (Page 2)**

- Format:
  arrayName.slice(*start*, [*end*])
  - *start* is inclusive; *end* is exclusive
- Examples:
  cars.slice(1, 3)
  - Returns the 2nd and 3rd elements

  cars.slice(2)
  - Returns the 2nd element to the end

  cars.slice(5, -1)
  - Returns the 6th element to the 2nd last

32 **The splice() Method (Page 1)**

- The Array.splice() method adds to and/or removes elements from a specific position in an array
- If removing elements, the method *returns* the removed items

33 **The splice() Method (Page 2)**

- Format:
  arrayName.splice(*index* [,*howMany*, [,*item1*, *item2*, ...]])
  - *index* is the location where the elements will added to or removed from (required)
  - *howMany* is the number of elements to be removed; must be zero (0) if elements are being added and not removed (optional)
  - The item1, item2, ... are the new elements being added (optional)

34 **The splice() Method (Page 3)**

- Example 1 (at *index* 2, add two items):
  var cars = ["Ford", "Chevy", "Honda", "Audi", "Mercedes"];

cars.splice(2, 0, "BMW", "Toyota");

- The *cars* array now will be ["Ford", "Chevy", "BMW", "Toyota", "Honda", "Audi", "Mercedes"]

35 **The splice() Method (Page 4)**

- Example 2 (at *index* 2, remove 2 items):

var cars = ["Ford", "Chevy", "BMW", "Toyota", "Honda", "Audi", "Mercedes"];
var temp = cars.splice(4, 2);

- The *cars* array now will be ["Ford", "Chevy", "BMW", "Toyota", "Mercedes"]
- The new *temp* array will be ["Honda", "Audi"]

36 **The splice() Method (Page 5)**

- Example 3 (at *index* 2, remove 1 item and then add two items:

var cars = ["Ford", "Chevy", "Honda", "Audi", "Mercedes"];
var temp = cars.splice(2, 1, "Lemon", "Kiwi");

- The *cars* array now will be ["Banana", "Orange", "Lemon", "Kiwi", "Mango"]
- The new *temp* array will be ["Apple"]

38 **Stack Processing with Arrays (Page 1)**

- The Array.push() *inserts* a new item to the *end* of an array
- The Array.pop() *deletes* an item from the *end* of an array and *returns* that item
- As such these methods together treat an array like a *stack* (last in, first out—LIFO)

39 **Stack Processing with Arrays (Page 2)**

- Formats:

arrayName.push(*item1*, *item2*, ...)
arrayName.pop()

- Examples:

cars.push("Ford", "Chevy", "Honda", "Audi");
var car = cars.pop();
document.write(cars.pop());

41 **The unshift() and shift() Methods (Page 1)**

- The Array.push() *inserts* a new item to the *front* of an array
  - The opposite of Array.push()
- The Array.pop() *deletes* an item from the *front* of an array and *returns* that item
  - The opposite of Array.pop()

42 **The unshift() and shift() Methods (Page 2)**

- Formats:

arrayName.unshift(*item1*, *item2*, ...)
arrayName.shift()

- Examples:

cars.unshift("Ford", "Chevy", "Honda", "Audi");

```
var car = cars.shift();
document.write( cars.shift() );
```

44 **The Array toString() Method (Page 1)**

- The Array.toString() method converts an array into a *single* string
- The returned string is a comma-delimited list
- The Array.toLocaleString() method converts an array into a string using local settings

45 **The Array toString() Method (Page 2)**

- Formats:
  arrayName.toString()
  arrayName.toLocaleString()
- Examples:
  document.write(cars.toString());
  document.write(cars.toLocaleString());
- The toString() method of an array object (actually any object) is called when the object name is referenced, e.g.:
  document.write(cars);

47 **The forEach() Method (Page 1)**

- The Array.forEach() method is an alternate method for iterating through an array
- It repeatedly executes a provided (named) function once per array element
- Format:
  arrayName.forEach(*namedFunction*);
- Example:
  cars.forEach(myFunction);

48 **The forEach() Method (Page 2)**

- The forEach() method passes up to *three arguments* to the named function
- Format:
  function *namedFunction(element*, *index* [, *array*])
  - *element*—the current element being processed from the array each time the function is called
  - *index*—the index of the current element
  - *array*—the entire array object itself
- Example:
  function myFunction(element, index) { …

50 **Strings as an Array of Characters (Page 1)**

- The String.charAt() gives the option of treating a string like an array in which each character is an element
- Format:
  string.charAt(*index*)

- Example:

```
for (var index = 0; index < car.length; index++)
{
   document.write(car.charAt(index);
}
```

52 **Strings as an Array of Characters (Page 2)**

- The String.split() method with no delimiter divides a string at each character to create an array of the individual characters
- Format:

  string.split()
- Example:

```
var chars = car.split();
```

54 **Arrays of Objects (Page 1)**

- An unlimited number of objects my be instantiated from a class
- Rather than declaring and instantiating and accessing the members of numerous objects, it is easier to create arrays of objects

55 **Arrays of Objects (Page 2)**

- Format to instantiate an array of objects from a class:

  var objectName = [];
- Example:

```
var cars = [];
```

56 **Arrays of Objects (Page 3)**

- Format to instantiate objects from an object array:

  objectName[*index*] = *objectReference*;
- Example to instantiate all objects in the cars[] array:

```
cars[0] =  createCar("Ford", 2009, "blue");
```

57 **Arrays of Objects (Page 4)**

- Format to call methods of an object array:

  objectName[*index*].*methodName*();
- Example to call methods of the cars[] array objects inside a for loop:

```
for (var index = 0; index < cars.length; index++)
{
   document.write( cars[index].getFullDescription() );
}
```

59 **Array Searches (Page 1)**

- Searching data involves determining whether a value (referred to as the target key or search key) is present in the data and, if so, finding its location.
- Two popular search algorithms are:

- The slower but simple linear array search
- The faster but more complex binary array search

60 **Array Searches (Page 2)**

- Searching algorithms all accomplish the same goal—finding the index of an element that matches the given target key, if such an element does, in fact, exist
- The major difference is the amount of effort they require to complete the search

61 **A Linear Array Search (Page 1)**

- The purpose of the linear search algorithm is to search array to *find the index* of the element in the array that matches the target key
- The target key is compared to *each element* in the array to see if a match exists

62 **A Linear Array Search (Page 2)**

- Once it is found, the index can be used to access elements in related arrays or objects
- If there are duplicate values in the array, linear search returns the index of the *first* element in the array that matches the search key

63 **A Linear Array Search (Page 3)**

- The linear search algorithm searches each element in an array *sequentially* (from beginning to end)
  - If target key is in array, algorithm tests each element until it finds the one that matches target key and returns the index of that element
  - If target key does not match an element in array, algorithm tests each element, and when the end of the array is reached, informs the user that the target key is not present

64 **A Linear Array Search (Page 4)**

- Format of a standard array search with a while loop:

```
var index = 0;

while (index < arrayName.length && targetKey != arrayName[index]
{
   index++;
}

if (index < arrayName.length)
{
   access the array element ...
}
```

65 **A Linear Array Search (Page 5)**

```
var car = prompt("Enter a car name", "");      // target key
```

```
var index = 0;

while (index < cars.length && car.toLowerCase() != cars[index].toLowerCase() )
{
   index++;
}

if (index < cars.length)
{
   document.write("<p>" + cars[index] + "</p>");
}
```

67 **A Linear Array Search with a for Loop**

```
var car = prompt("Enter a car name", "");

for (var index = 0;
      index < cars.length && car.toLowerCase() != cars[index].toLowerCase();
      index++);

if (index < cars.length)
{
   document.write("<p>" + cars[index] + "</p>");
}
```

69 **Multiple-Dimensional Arrays (Page 1)**

- An array with two or more indexes
- In a 2-dimensional array, the two indexes are conceptualized as a row index and a column index

70 **Multiple-Dimensional Arrays**

71 **Multiple-Dimensional Arrays (Page 2)**

- Format (declaring a 2-dimensional array)
  var *arrayName* = [][];
- Example:
  var cars = [][];

72 **Multiple-Dimensional Arrays (Page 3)**

- There can be as many levels of dimension as the programmer desires
- Example (4-dimensional array):
  var myArray = [][][][];
  - The more brackets used, the deeper the hole dug

73 **Declaring a 2-dimensional Array**

- Example of instantiating and initializing a 2-dimensional array using array literals:

```
var cars =
[
   ["Ford", 2009, "blue"],
   ["Chevy", 2015, "grey"],
   ["Honda", 2010, "red"]
];
```

74 **Iterating through a 2-dimensional Array (Page 1)**

- Accessing all of the elements of the two index levels of the 2-dimensional array involves the use of a *nested* loop
- Format:

```
for (index1 = 0; index1 < index1Length; index1++)
{
    for (index2 = 0; index2 < index2Length; index2++)
    {
        … arrayName[index1][index2] …
    }
}
```

75 **Iterating through a 2-dimensional Array (Page 2)**

- Example:

```
for (index1 = 0; index1 < index1Length; index1++)
{
    for (index2 = 0; index2 < index2Length; index2++)
    {
        document.write(cars[index1][index2]);
    }
}
```