3416 JavaScript and AngularJS

Thank you!

Thank you for choosing the University of Toronto School of Continuing Studies

Our Courses & Programs

The School offers more than 600 courses in 80 certificates, both classroom-based and online, covering a vast range of interests and specializations:

- Business & Professional Studies
- English Language Program
- Arts & Science
- Languages & Translation
- Creative Writing

Follow us on social

Join the conversation with us online:

facebook.com/UofTLearnMore

@UofTLearnMore

linkedin.com/company/university-of-toronto-school-of-continuing-studies

Module 3

INTRODUCTION TO JAVASCRIPT

Module 2: Learning Outcomes

- Passing Variables by Value vs by Reference, Function Constructors, Object literals and the this keyword, and DOM Manipulation

Passing Variables by Value vs by Reference

- Given b=a, passing/copying by value means changing copied value in b does not affect the value stored in a vice versa.
- Given b=a, passing/copying by reference means changing copied value in b does affect the value stored in a and vice versa.
- In Javascript, primitives are passed by value, objects are passed by reference.

Passing Variables by Value vs by Reference

- Primitives

```
var a = 7;
var b = a;
```

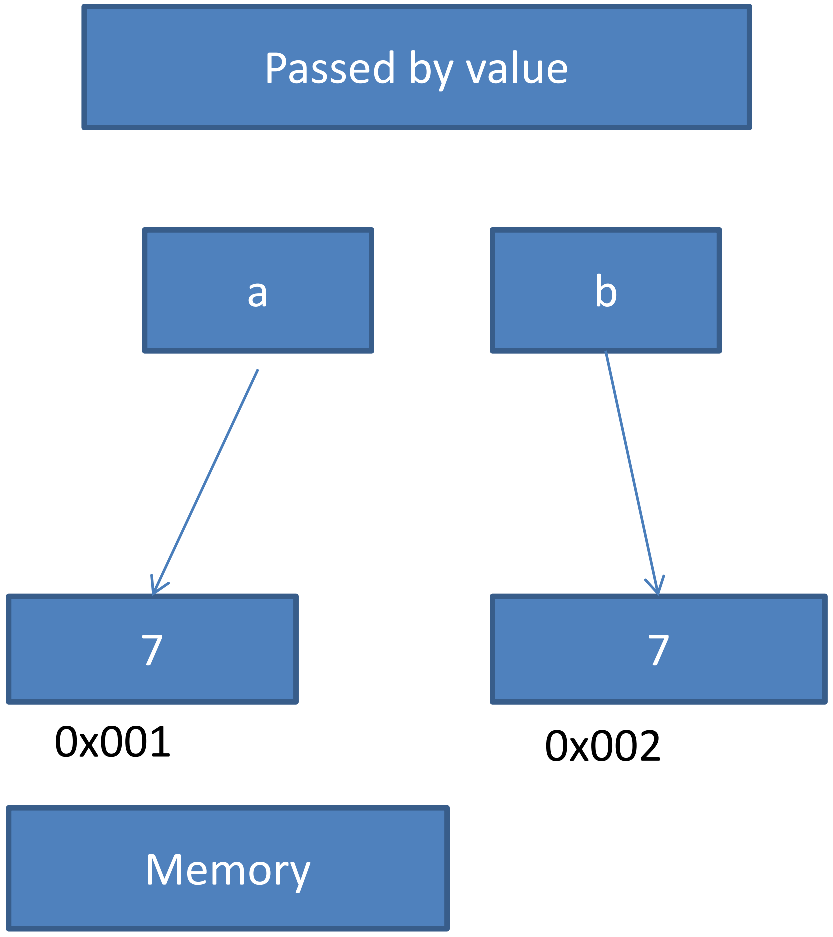
objects

```
var a = {x: 7};
var b = a;
```

‘b’ ends up with the same value as ‘a’ in both circumstances.

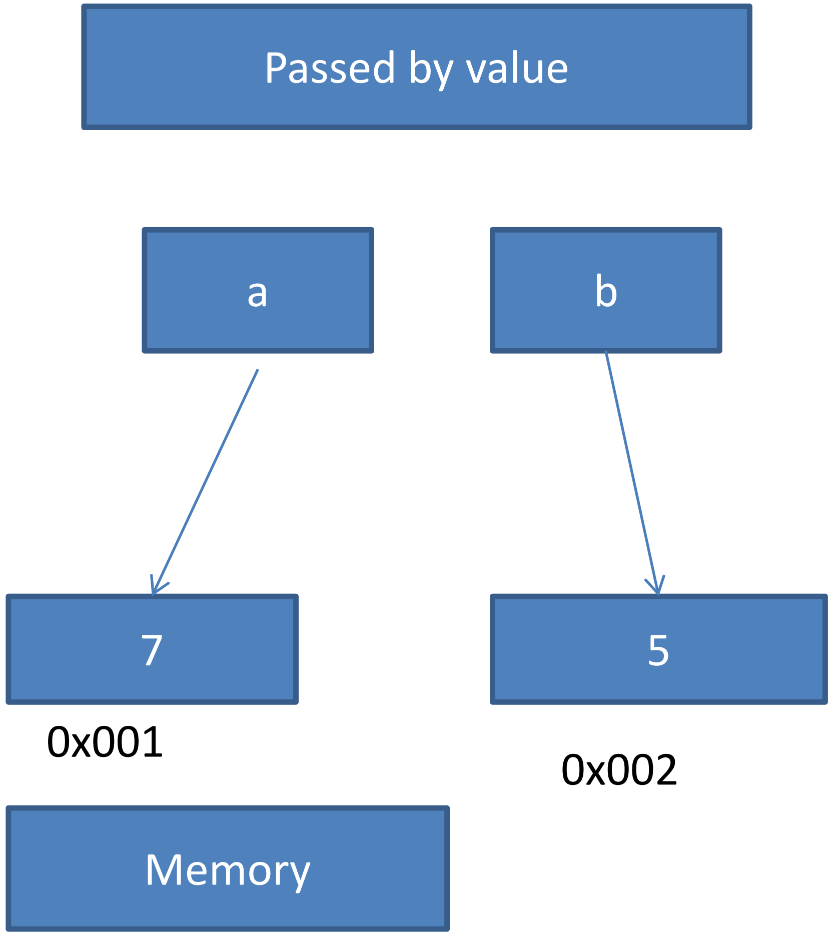
How does that work?

Passed by value



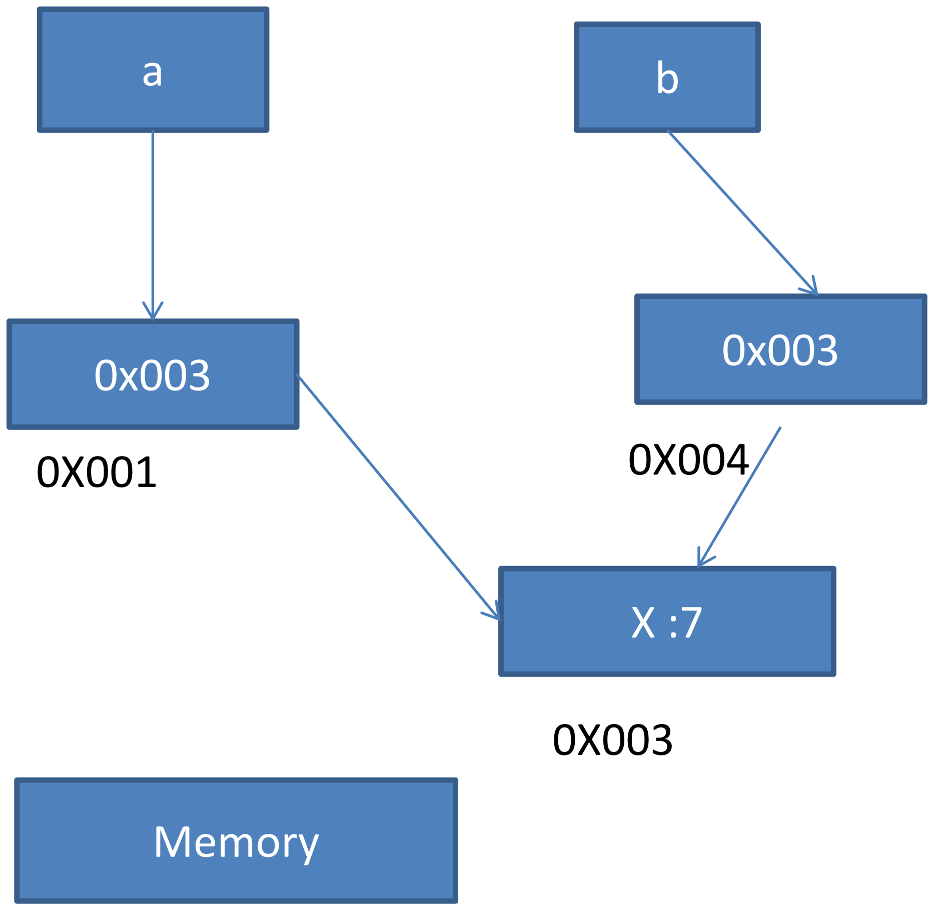
```
var a = 7;
var b = a;
```

Passed by value or copy by value



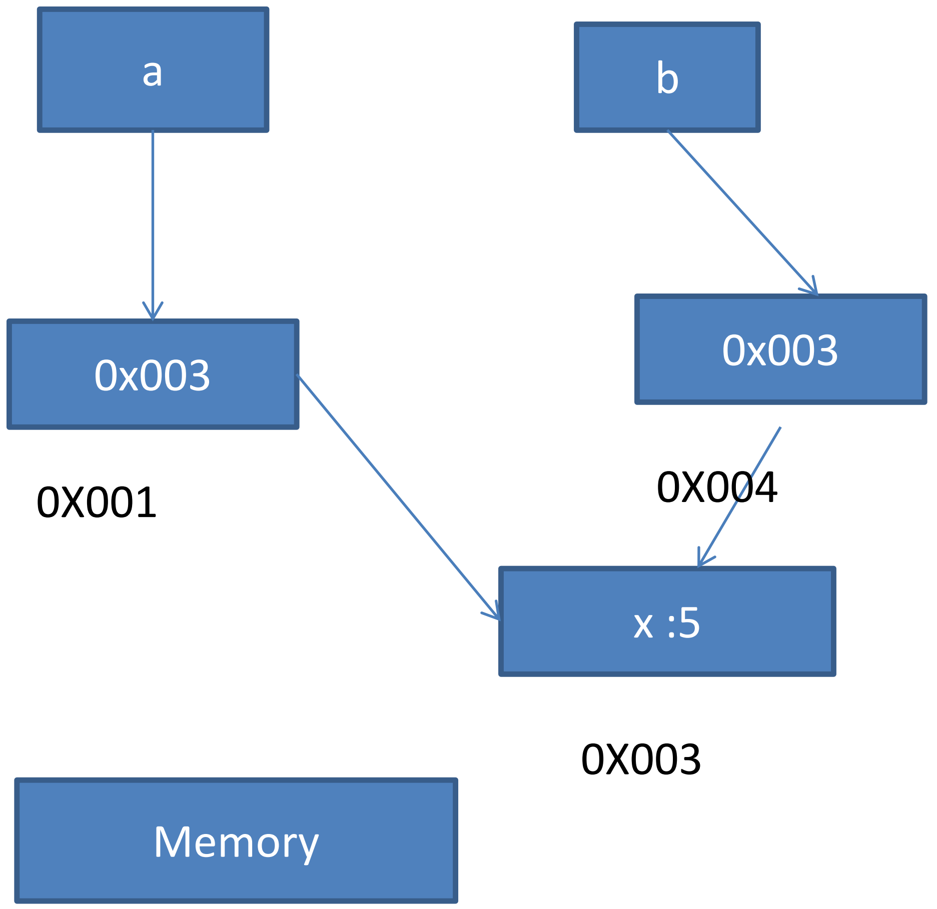
```
var a = 7;
var b = a;
B=5;
```

Passed by reference



```
var a = {x:7};
var b = a;
```

Passed by reference



```
var a = {x:7};
var b = a;
b.x=5;
```

Pass by value vs Pass By Reference

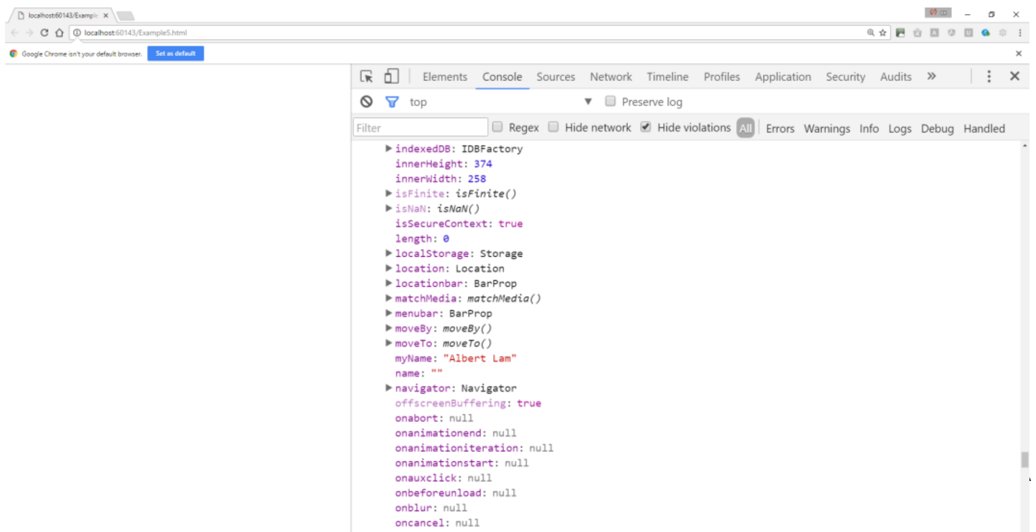
- Difference between pass by value and pass by reference is that pass by value the value once is changed doesn’t affect original variable. As opposed as passed by reference when you change the value and the object that is passed by reference the original object that is pointing to it is changed as well.
- Create Example1.html,Example2.html and Example3.html and Example4.html

Pass by value vs Pass By Reference

- Simple rule to remember
  - Primitives are copied by value
  - Object are passed by reference

Function Constructors Prototype(Inheritance) and the this keyword

- Create Example5.html , Example6.html, Example7.html



```
▶ indexedDB: IDBFactory
  innerHeight: 374
  innerWidth: 258
▶ isFinite: isFinite()
▶ isNaN: isNaN()
  isSecureContext: true
  length: 0
▶ localStorage: Storage
▶ location: Location
▶ locationbar: BarProp
▶ matchMedia: matchMedia()
▶ menubar: BarProp
▶ moveBy: moveBy()
▶ moveTo: moveTo()
  myName: "Albert Lam"
  name: ""
▶ navigator: Navigator
  offscreenBuffering: true
  onabort: null
  onanimationend: null
  onanimationiteration: null
  onanimationstart: null
  onauxclick: null
  onbeforeunload: null
  onblur: null
  oncancel: null
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Javascript Object Prototypes

- Every Javascript Object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype.
- Objects created using an object literal, or with new Object (), inherit from a prototype called Object.prototype.
- Example8A.html

Javascript Object Prototypes

- Adding a Properties and Methods to Objects
  - Sometimes you want to add new properties (or methods) to an existing object.
  - Example8B.html (The property will be added to myFather. Not to myMother. Not to any other person objects.)
- Adding a Method to an Object
  - Adding a new method to an existing object .
  - Example8C.html

Javascript Object Prototypes

- Adding Properties to a Prototype
- You cannot add a new property to a prototype the same way as you add a new property to an existing object, because the prototype is not an existing object.

- Example8D.html
- To add a new property to a prototype, you must add it to the constructor function:
- Example8E.html
- Adding Methods to a Prototype
- Your constructor function can also define methods:
- Example8F.html
- Using the **prototype** Property
- The JavaScript prototype property allows you to add new properties to an existing prototype:
- Example8G.html
- The JavaScript prototype property also allows you to add new methods to an existing prototype:
- Example8h.html

- Example8I.html,Example9.html, Example10.html ,Example10A.html,Example10B.html,Example10C.html,Example10D.html, and Example11.html

Object literals and the this keyword

- Created Eample12.html, Example13.html, Example13A.html,Example14.html

Javascript Arrays

- JavaScript arrays are used to store multiple values in a single variable.
- Create Example15.html
- What is an Array?
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.
-

Creating an Array

- Using an array literal is the easiest way to create a JavaScript Array.
- Syntax:
- var *array_name* = [*item1*, *item2*, ...];
- Create Example16.html
- Spaces and line breaks are not important. A declaration can span multiple lines:
- Create Example17.html
-

Using the JavaScript Keyword new

- The following example18.html also creates an Array, and assigns values to it:
- Example17.html and Example18.html do exactly the same. There is no need to use new Array().
  For simplicity, readability and execution speed, use the first one (the array literal method).

Access the Elements of an Array

- You refer to an array element by referring to the **index number**.
- This statement accesses the value of the first element in cars:
- var name = cars[0];
- This statement modifies the first element in cars:
- cars[0] = "Opel";
- Create Example19.html

Access the Full Array

- With JavaScript, the full array can be accessed by referring to the array name:
- Create Example20.html

Arrays are Objects

- Arrays are a special type of objects.
  The **typeof** operator in JavaScript returns "object" for arrays.
- But, JavaScript arrays are best described as arrays.
- Arrays use **numbers** to access its "elements". In this example, **person[0]** returns John:
- Array:
- var person = ["John", "Doe", 46];

- Create Example21.html

Arrays

- Objects use **names** to access its "members". In this example, **person.firstName** returns John:
- Object:
  - var person = {firstName:"John", lastName:"Doe", age:46};
- Created Example22.html

Array Elements Can Be Objects

- JavaScript variables can be objects. Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:
- myArray[0] = Date.now;
  myArray[1] = myFunction;
  myArray[2] = myCars;

Array Properties and Methods

- The real strength of JavaScript arrays are the built-in array properties and methods:
- Examples
- var x = cars.length;   // The length property returns the number of elements
  var y = cars.sort();   // The sort() method sorts arrays

The length Property

- The **length** property of an array returns the length of an array (the number of array elements).
- Example
- var fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.length;   // the length of fruits is 4
- Created Example23.html
- The length property is always one more than the highest array index.

Looping Array Elements

- The best way to loop through an array, is using a "for" loop:
- Example
- var fruits, text, fLen, i;

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i < fLen; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
```

- Created Example24.html

Adding Array Elements

- The easiest way to add a new element to an array is using the push method:
- Created Example25.html
- New element can also be added to an array using the length property:
- Created Example26.html
- **WARNING !**
- Adding elements with high indexes can create undefined "holes" in an array:
- Created Example27.html

Associative Arrays (Named Indexes)

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does **not** support arrays with named indexes.
- In JavaScript, **arrays** always use **numbered indexes**.
- Create Example28.html

Associative Arrays (Named Indexes)

- **WARNING !!**
  If you use named indexes, JavaScript will redefine the array to a standard object.
  After that, some array methods and properties will produce **incorrect results**.
- Example:
- var person = [];
  person["firstName"] = "John";
  person["lastName"] = "Doe";
  person["age"] = 46;
  var x = person.length;   // person.length will return 0
  var y = person[0];   // person[0] will return undefined
- Create Example29.html

Two ways to create Arrays

- Create Example30.html

Avoid new Array()

- Created Example31.html

How to Recognize an Array

- A common question is: How do I know if a variable is an array?
- The problem is that the JavaScript operator **typeof** returns "object":
- The typeof operator returns object because a JavaScript array is an object.
- Created Example32.html

Two ways to recognize an array

- Solution 1:
- To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:
- Array.isArray(fruits);   // returns true
- Created Example33.html
- Solution 2:
- The **instanceof** operator returns true if an object is created by a given constructor:
- var fruits = ["Banana", "Orange", "Apple", "Mango"];

  fruits instanceof Array   // returns true
- Created Example34.html

JavaScript Array Methods

- Converting Arrays to Strings
- The JavaScript method **toString()** converts an array to a string of (comma separated) array values.
- Created Example35.html
- The **join()** method also joins all array elements into a string.
- It behaves just like toString(), but in addition you can specify the separator:
- Created Example36.html
- Do Exercise1Question.html,Exercise2Question.html,Exercise3Question.html,Exercise4Question.html, and Exercise5Question.html

Popping and Pushing

- When you work with arrays, it is easy to remove elements and add new elements.
- This is what popping and pushing is:
- Popping items **out** of an array, or pushing items **into** an array.
- Popping
- The **pop()** method removes the last element from an array:
- The pop() method returns the value that was "popped out":
- Created Example37.html

Pushing

- The **push()** method adds a new element to an array (at the end):
- The push() method returns the new array length:
- Created Example38.html

Shifting Elements

- Shifting is equivalent to popping, working on the first element instead of the last.
- The **shift()** method removes the first array element and "shifts" all other elements to a lower index.
- The shift() method returns the string that was "shifted out":
- Create Example39.html and Example40.html

unshift() method

- The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:
- The unshift() method returns the new array length.
- Created Example41.html

Changing Elements

- Array elements are accessed using their **index number**:
- Array **indexes** start with 0. [0] is the first array element, [1] is the second, [2] is the third ...
- Create Example42.html
- The length property provides an easy way to append a new element to an array:
- Create Example43.html

Deleting Elements

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**:
- Using **delete** may leave undefined holes in the array. Use pop() or shift() instead.
- Create Example44.html
-

Splicing an Array

- The **splice()** method can be used to add new items to an array:
- Example:
- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.splice(2, 0, "Lemon", "Kiwi");
- The first parameter (2) defines the position **where** new elements should be **added** (spliced in).
- The second parameter (0) defines **how many** elements should be **removed**.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.
- Create Example45.html and Example46.html

Using splice() to Remove Elements

- With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:
- Example:
- var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);
- // Removes the first element of fruits
- The first parameter (0) defines the position where new elements should be **added** (spliced in).
- The second parameter (1) defines **how many** elements should be **removed**.
- The rest of the parameters are omitted. No new elements will be added.
- Created Example47.html

Merging (Concatenating) Arrays

- The **concat()** method creates a new array by merging (concatenating) existing arrays:
- Example (Merging Two Arrays)
- var myGirls = ["Cecilie", "Lone"];
  var myBoys = ["Emil", "Tobias","Linus"];
  var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
- The concat() method does not change the existing arrays. It always returns a new array.
- Created Example48.html

Merging (Concatenating) Arrays

- The concat() method can take any number of array arguments:
- Created Example49.html
- The concat() method can also take values as arguments:
- Example (Merging an Array with Values)
- var arr1 = ["Cecilie", "Lone"];
  var myChildren =
  arr1.concat(["Emil", "Tobias","Linus"]);
- Created Example50.html

Slicing an Array

- The **slice()** method slices out a piece of an array into a new array.
- This example slices out a part of an array starting from array element 1 ("Orange"):
- Example
- var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
  var citrus = fruits.slice(1);
- The slice() method creates a new array. It does not remove any elements from the source array.
- Create Example51.html

Slicing an Array

- This example slices out a part of an array starting from array element 3 ("Apple"):
- Example
- var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
  var citrus = fruits.slice(3);
- Created Example52.html

Slicing an Array

- The slice() method can take two arguments like slice(1, 3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.
- Example
- var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
  var citrus = fruits.slice(1, 3);
- Created Example53.html

Slicing an Array

- If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.
- Example
- var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
  var citrus = fruits.slice(2);
- Created Example54.html

Automatic toString()

- JavaScript automatically converts an array to a comma separated string when a primitive value is expected.
- This is always the case when you try to output an array.
- These two examples will produce the same result:
- Created Example55.html and Example56.html
- Do Exercise6Question.html,Exercise7Question.html, Exercise8Question.html,Exercise9Question.html

Arrays

- Arrays are a collection of data.

- Create Example57.html, Example58.html,
- Example59.html, Example60.html, Example61.html

DOM Manipulation

- DOM(Document Object Model) Manipulation
- Create Example62.html, and Example63.html