Arrays

An array is created by assigning an array literal to a variable

```javascript
var arrayVar = [2,3,6,7,11];
```

Operators on Array Literals

- The length operation returns the number of elements in an array literal: `arrayLiteral.length`

```javascript
[2,3,6,7,11].length // returns 5
```

Array Literals

- An array literal is a comma-separated list of values enclosed in square brackets

```javascript
[2,3,6,7,11]
```

- Each element of that list has an index position given by the number of other elements preceding it:

```javascript
[2,3,6,7,11]
```

Arrays Definition

- Arrays have no fixed length and it is always possible to add more elements to an array

```javascript
var arrayVar = [2,3,6,7,11];
```

- Accessing an element of an array that has not been assigned a value yet

```javascript
arrayVar[0]
```

Arrays Iteration: for-loop and forEach-method

- The recommended way to iterate over all elements of an array is a for-loop

```javascript
for (let i = 0; i < arrayVar.length; i++) {
    // arrayVar[i] = new value
}
```

Storing an array variable's length

```javascript
arrayVar.length;
```

Assigning an array to a new variable creates a reference to the original array:

```javascript
var newVar = arrayVar;
```

- Arrays are also passed to functions by reference

```javascript
function printMyArray(arr) {
    console.log(arr);
}
printMyArray(arrayVar);
```

- The slice() function can be used to create a proper copy of an array:

```javascript
array.slice(start, end)
```

- Returns a copy of those elements of array variable that have indices between start and end

```javascript
array.slice(0, 4);
```

For a nested array literal, it is possible to iterate this access operation

```javascript
var arrayLiteral = [2, [3, 5], [7, [11]]];
```

```javascript
arrayLiteral[1][0] // returns 3
```

Assigning an array to a new variable creates a reference to the original array:

```javascript
arrayVar = arrayLiteral
```

```javascript
arrayVar[3] = 7
```

```javascript
arrayVar[0] = [Jackson, Dave]
```

```javascript
arrayVar[1][0] = Hustadt, Ullrich
```

```javascript
arrayVar[2][3] = 7
```

```javascript
arrayVar[3] = undefined
```

```javascript
arrayVar[4] = undefined
```

```javascript
arrayVar[5] = world
```

```javascript
arrayVar.length = 4
```

```javascript
console.log(arrayVar[0]); // returns [Jackson, Dave]
```

```javascript
console.log(arrayVar[1]); // returns Hustadt, Ullrich
```

```javascript
console.log(arrayVar[2]); // returns undefined
```

```javascript
console.log(arrayVar[3]); // returns undefined
```

```javascript
console.log(arrayVar[4]); // returns undefined
```

```javascript
console.log(arrayVar[5]); // returns world
```

```javascript
console.log(arrayVar.length); // returns 4
```

```javascript
var arrayLiteral = [2, [3, 5], [7, [11]]];
```

```javascript
var arrayLiteral = [2, [3, 5], [7, [11]]][1][0]; // returns 3
```

```javascript
var arrayLiteral = [2, [3, 5], [7, [11]]][1]; // returns [3, 5]
```

```javascript
var arrayLiteral = [2, [3, 5], [7, [11]]][1][0]; // returns 3
```

Arrays Iteration: for-loop and forEach-method

```javascript
// Example: Array Iteration
var arrayVar = ['hello', [1, 2], function () { return 5}, 43];
```

```javascript
arrayVar.forEach(rewriteNames);
```

```javascript
var rewriteNames = function (elem, index, arr) {
    console.log('myArray['+index+'] = '+ myArray[index]);
    arr[index] = elem.replace(/\s\$/g, '').toLowerCase();
}
```

```javascript
myArray = ['Dave Jackson', 'Ullrich Hustadt'];
```

```javascript
myArray.forEach(rewriteNames);
```

```javascript
for (let i = 0; i < myArray.length; i++) {
    console.log(myArray[i] + 1);
}
```

```javascript
{arr[index] = elem.replace(/\s\$/g, "\$2, \$1");}
```

```javascript
var myArray[0] = Jackson, Dave
myArray[1] = Hustadt, Ullrich
```

```javascript
var myArray[0] = Jackson, Dave
myArray[1] = Hustadt, Ullrich
```

- The forEach method can be used to create a proper copy of an array:

```javascript
array.slice(start, end)
```

- Returns a copy of those elements of array variable that have indices between start and end

```javascript
array.slice(0, 4);
```
Stacks and Queues

Arrays and Functions: Example

```javascript
function bubble_sort(array) {
  if (!((array && array.constructor == Array)))
    throw("Argument not an array")
  for (var i=0; i<array.length -1; i++) {
    if (array[i] < array[i+1]) {
      // swap can change array because array is
      // passed by reference
      swap(array, i, i+1)
    }
  }
  return array
}

function swap(array, i, j) {
  let tmp = array[i]
  array[i] = array[j];
  array[j] = tmp;
}
```

Array functions

JavaScript has no stack or queue data structures, but has stack and queue functions for arrays:

- ```array.push(value1, value2, ...)``` appends one or more elements at the end of an array (enqueue); returns the number of elements in the resulting array
- ```array.pop()``` extracts the last element from an array and returns it
- ```array.shift()``` extracts the first element of an array (dequeue) and returns it
- ```array.unshift(value1, value2, ...)``` inserts one or more elements at the start of an array variable; returns the number of elements in the resulting array

Equality and Program Behaviour

Why do we care whether 5 == true is true or false?
~ it influences whether more complex objects are equal or not
~ it influences whether more complex objects are equal or not

JavaScript:

```javascript
if (5 == true) {
  document.writeln("5 is equal to true");
} else {
  document.writeln("5 is not equal to true");
}
```

Output: 5 is true and 5 is equal to true

PHP:

```php
if (5) {
  print("5 is true");
} else {
  print("5 is not true");
}
```

Output: 5 is true and 5 is not equal to true

Stacks and Queues

Stack
A collection of items that are inserted and removed according to the last-in-first-out (LIFO) principle
- push adds an item to the top of the stack
- pop removes the top item from the stack

Queue
A collection of items that are inserted and removed according to the first-in-first-out (FIFO) principle
- enqueue adds an item to the back of the queue
- dequeue removes the item at the front of the queue
Equality and Program Behaviour

Note: The way in which more complex data structures are compared also differs between PHP and JavaScript.

JavaScript:
```javascript
array3 = ['1.23e2', 5];
array4 = ['1.23e1', true];
if (array3 == array4)
    console.log('The two arrays are equal');
else console.log('The two arrays are not equal');
```
Output: The two arrays are equal

PHP:
```php
$array3 = array('1.23e2', 5);
$array4 = array('1.23e1', true);
if ($array3 == $array4)
    print('The two arrays are equal');
else print('The two arrays are not equal');
```
Output: The two arrays are equal

Revision and Further Reading


JavaScript Libraries: Example

```javascript
function bubble_sort(array) {
    function swap(i, j) {
        // swap elements
    }
    for (let i = 0; i < array.length - 1; i++) {
        for (let j = 0; j < array.length - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                swap(j, j + 1);
            }
        }
    }
    return array;
}
```

Example:
```html
<html lang="en-GB">
<head>
    <title>Sorting example</title>
    <script src="http://cgi.csc.liv.ac.uk/~ullrich/jsLib.js"></script>
</head>
<body>
    <script>
        array = [20, 4, 3, 9, 6, 8, 5, 10];
        sorted = bubble_sort(array.slice(0));
    </script>
</body>
</html>
```

JavaScript Libraries: Import Statements

• Collections of JavaScript functions (and other code), libraries, can be stored in one or more files and then be reused
• By convention, files containing a JavaScript library are given the file name extension .js
• `<script>` tags are not allowed to occur in the file
• A JavaScript library is imported using
  ```javascript
  <script src="url"></script>
  ```
  where `url` is the (relative or absolute) URL for library

One such import statement is required for each library

• Import statements are typically placed in the head section of a page or at the end of the body section (see next slide)
• Web browsers typically cache libraries

• JavaScript code may change the HTML markup of an HTML document
• Consequently, whenever a browser encounters a script element, by default, it stops parsing the remaining HTML markup of the page until that script element has been processed

~ poor user experience

• ‘Safe solution’: Put script elements at the end of the body element
• ‘Better solution’: Use the async or defer attribute of the script element to change the default behaviour

```javascript
<script src="jsLib1.js" async></script>
<script src="jsLib2.js" defer></script>
```
Do not wait for the processing of the script elements, fetch and execute `jsLib1.js` and `jsLib2.js` (in parallel) in any order

```javascript
<script src="jsLib1.js" defer></script>
<script src="jsLib2.js" defer></script>
```
Do not wait, fetch `jsLib1.js` and `jsLib2.js` (in parallel), execute in order after parsing is finished