



# PIC 40A

## Lecture 11: JS: Functions, events, arrays

# JavaScript Functions

Syntax:

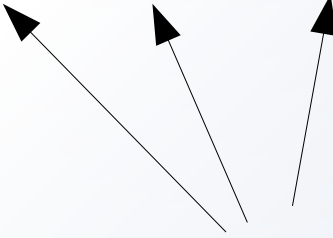
```
function function_name(p1, p2, ..., pn)  
{
```

```
// Some code;
```

```
return some_value; // optional
```

```
}
```

Optional parameters



# Couple of things to note

---

The return values type is implicitly defined.

There may be several return statements in a function, each returning data of different type.

Variables are passed by value. There is no pass by reference for ordinary variables. (Objects are another story...)

There is no type checking of parameters

# Where do we place them?

---

- Functions are placed in the head section
- Functions are not automatically executed when the page loads.
- A function must be called somehow.

# Example

```
<head>
  <script type="text/javascript">
    function bottles_beer(num)
    {
      for (var i = num; i > 0; i--)
      {
        document.write(i + "bottles of
beer...<br/>");
      }
    }
  </script>
</head>
```

# How can we call a function?

Example:

```
<script type="text/javascript">  
  var number = prompt("How many bottles?");  
  bottles_beer(number);  
</script>
```

# How can we call a function?

---

Functions can call functions.

Example:

```
function foo()  
{  
    alert("Whats up?");  
}
```



# JavaScript onclick event

---

Most often a function call is triggered by some event such as user clicking on a button.

```
<form action="" method="get">  
  <input type="button" value="button!"  
    onclick="alert('Hey quit clicking me!')" />  
</form>
```



# Another cool way to call a function

setTimeout can be used to execute JS after a given time interval.

Syntax:

```
var e = setTimeout("javascript statement",milliseconds);
```

See example.

Recall that 1000ms=1sec.

Similar command is

```
var e = setInterval("javascript statement",milliseconds);
```

This repeats the JS after the interval.

# Canceling timed events

---

To cancel the repeating use `clearInterval`

Syntax:

```
clearInterval(id_of_the_repeater);
```

See example.

# Global vs local

## Variable scope

In JavaScript, variable scope can be *global* or *local*.

### Global

A variable that is declared outside any functions is *global*. This means it can be referenced anywhere in the current document.

- Declared outside any functions, with or without the **var** keyword.
- Declared inside a function without using the **var** keyword, but only once the function is called.

# Global vs local

## Local

A variable that is declared inside a function with keyword **var** is *local*. This means it can only be referenced within the function it is declared.

## Other notes

If a variable is declared inside a conditional statement, it is still available anywhere following the declaration in the containing function (or globally if the conditional is not in a function).

However, it will equal *undefined* if the condition evaluates to false, unless the variable is later assigned a value. If a local variable is referenced globally or in another function, a JavaScript error will occur.

# Example

```
var numberCars = 3; // global
numberTrees = 15; // global
if (numberTrees > numberCars)
{
    var numberRoads = 4; // global
}
else
{
    var numberLakes = 9; // global, but will be undefined since never get in here.
}

function simpleFunction()
{
    var colorCar = 'blue'; // local
    colorTree = 'green'; // global, once this function is called
    if (colorCar !== colorTree)
    {
        var colorRoad = 'grey'; // local anywhere in this function after this line
    }
    else
    {
        var colorLake = 'aqua'; // local, but will be undefined since never get in here.
    }
}
```

# JS Arrays

---

- An array is an object
- Contains data elements in sequential order
- Elements need **not** be of the same type
- Elements can be primitive values or object references (possibly functions or other arrays)
- Has dynamic length

# JS arrays

## Created via the **Array constructor**:

// creates an empty array of length 3

```
var a1= new Array(3);
```

// creates an array of length 4

```
var a2 = new Array("7", 1, new Date(), true);
```

## Created via an **Array literal**:

```
var a3 = ["7", 1, new Date(), false];
```



# Filling an Array

```
var a1 = new Array();  
a1[0] = 27;  
a1[45] = "Hello";
```

Arrays length is the highest slot filled + 1

All array elements are allocated dynamically on the heap

See `insert_names.js` for an example with for loops and arrays

# Important JS array property

---

JS arrays have a length property:

```
var a3 = ["7", 1, new Date(), false];
```

```
var len = ar3.length; // len now has value  
4
```

# Array Object Methods

- `join(delimiter)`
  - converts all elements to strings and then concatenates them to a single string, separated by given delimiter
- `reverse()`
  - reverses order of elements
- `sort()`
  - sorts array elements as strings, not numbers
- `concat(a1, a2, ...)`
  - returns a new array that joins the given arrays to the calling array

# Array Object Methods

`toString()`

- Elements turn to strings and are concatenated separated by commas

`push(e)`

- Add element `e` to end

`pop()`

- Remove element from end

`unshift(e)`

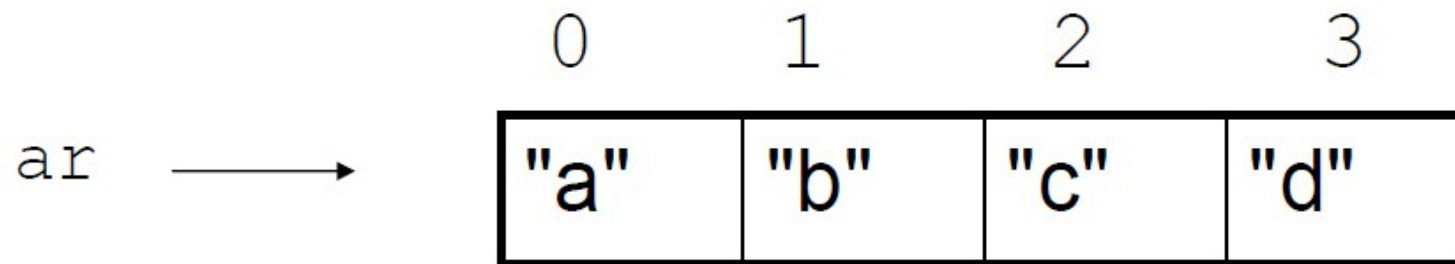
- Add element `e` to beginning

`shift()`

- Remove element from beginning

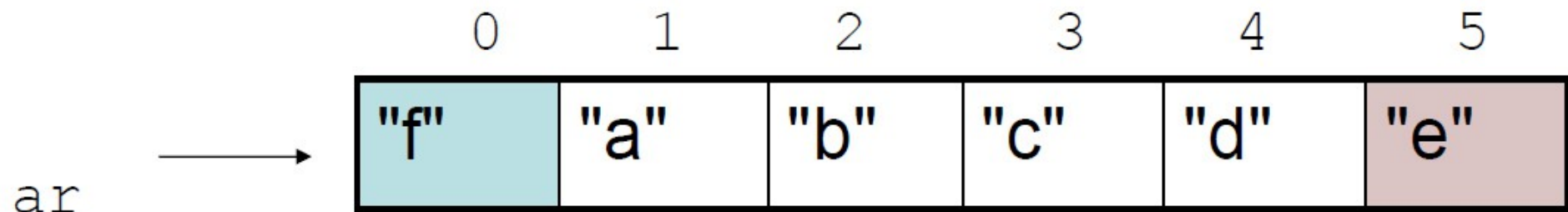
# Array Example

```
var ar = ["a", "b", "c", "d"];
```



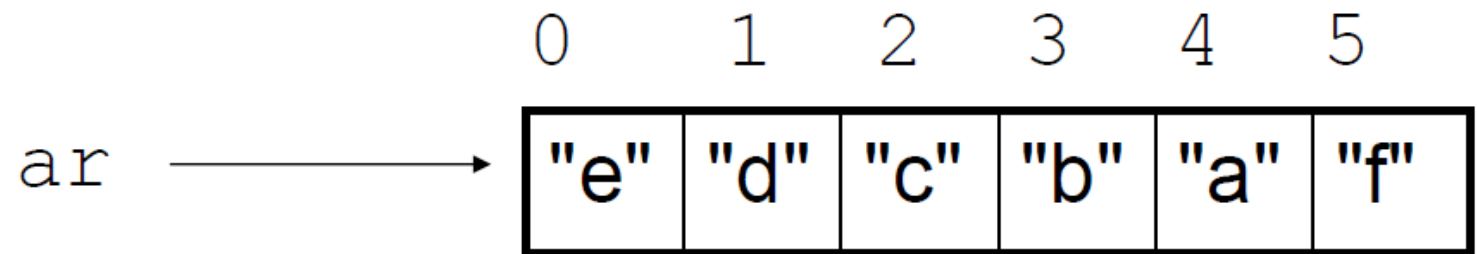
```
ar.push("e");
```

```
ar.unshift("f");
```

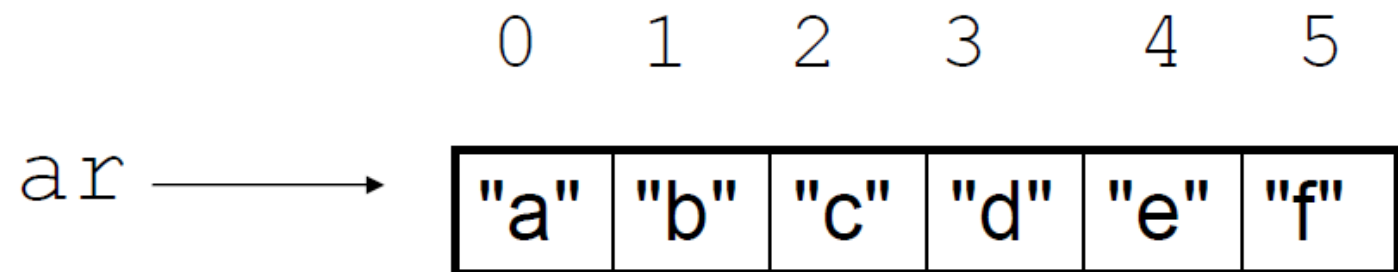


# Array Example

```
ar.reverse();
```

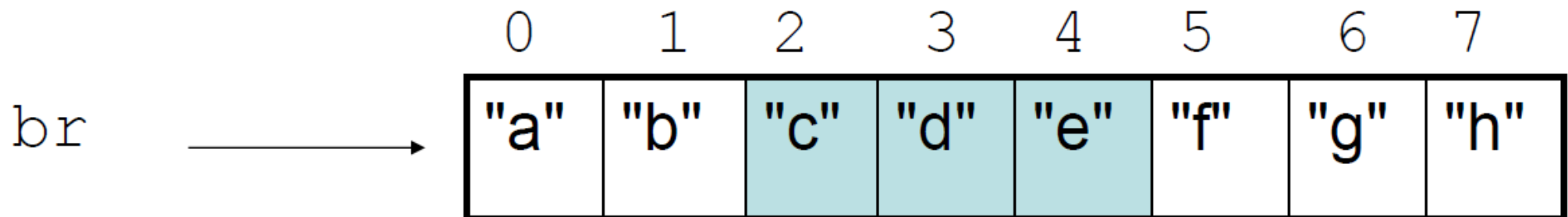


```
ar.sort();
```

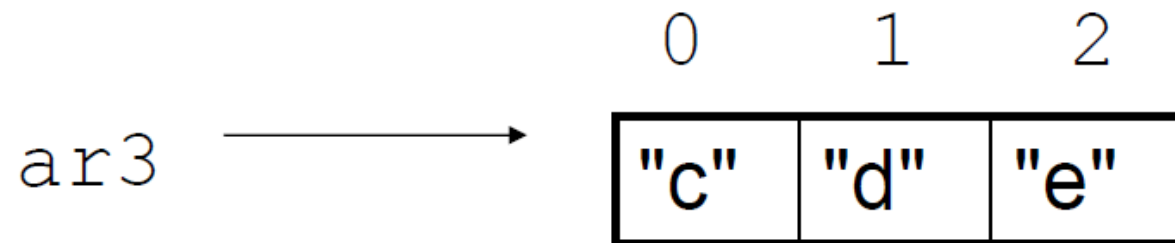


# Array Example

```
var ar2 = ["g", "h"];  
var br = ar.concat(ar2);
```



```
var ar3 = br.slice(2, 5);
```





# Array example

`ar.toString()` has value

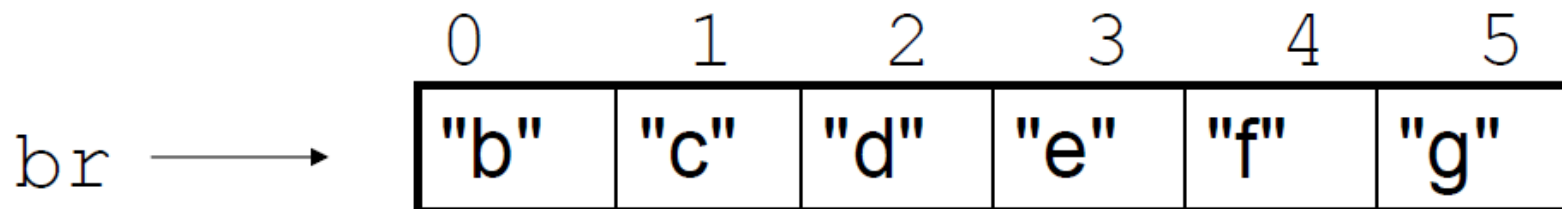
`"a,b,c,d,e,f,g,h"`

`ar.join("$")` has value

`"a$b$c$d$e$f$g$h"`

# Array Example

```
var first_letter = br.shift();  
var last_letter = br.pop();
```



first\_letter

"a"
-----

last\_letter

"h"
-----

# Useful string method

split can be used to take a string and make an array out of it.

Syntax:

```
str.split(delimiter);
```

Example:

```
var str = "H,e,l,l,o";  
var ar = str.split(",");
```

ar is now an array that looks like this:

H	e	l	l	o
---	---	---	---	---

# Function literals

In JavaScript variables can store functions!

We can define a function using a **function literal**

```
var f = function(message) {  
  alert(message);  
};
```

```
f("Hello!");           // calling the function
```

# Example:

```
var f = function(message) {  
  alert(message);  
};
```

```
f("hello");           // calling the function  
var a = [f, 2];        // storing function f in  
                        // an array a  
a[0]("there");         // using array a to call  
                        // function f
```

# Example

You can place function literals in an array:

```
var a =  
[  
    function(x,y){ return (x+y); },  
    function(x,y){ return (x-y); }  
];
```

```
var sum = a[0](1,2); //3  
var difference = a[1](1,2); //-1
```