

Scripting and Web Applications EE1081

Lecture 3 JavaScript

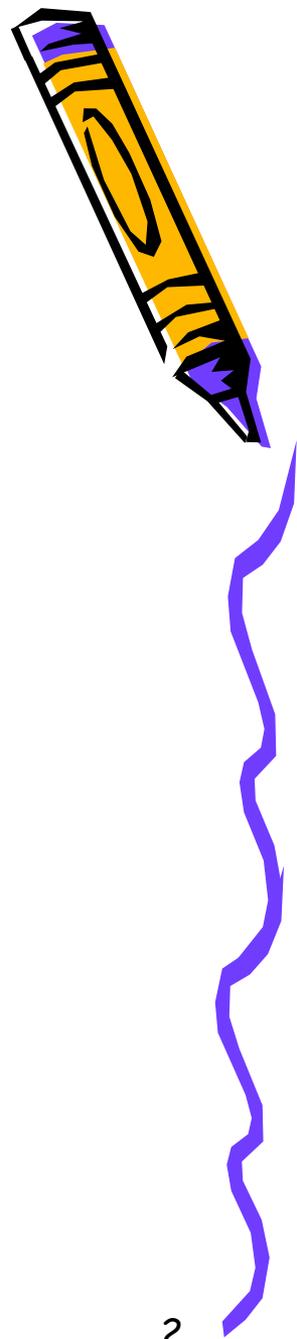
By: A. Mousavi

SERG, School of Engineering Design, Brunel University, UK



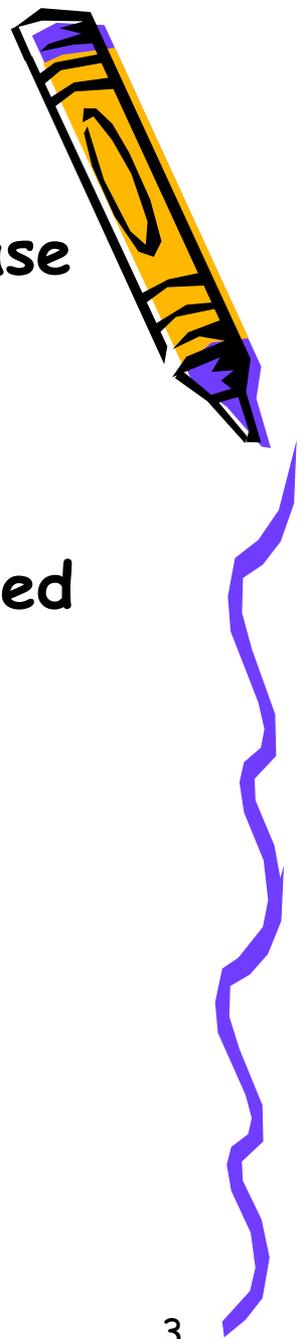
JavaScript Syntax Cont.

1. Conditional statements
2. Looping statements
3. Functions
4. Objects



Conditional statements

- In many real world cases you may want to base your decision on a specified criterion
- For example a condition needs to be met for something to happen (e.g. if you are registered for this course you can view the course information)
- So you need **conditional statements**



Conditional statements cont.

- When the browser interprets the JavaScript, it executes the statements one after the other
- First set the condition to evaluate whether it is met, then the execution of other statements can follow

```
if(condition) {  
    statement;  
}
```



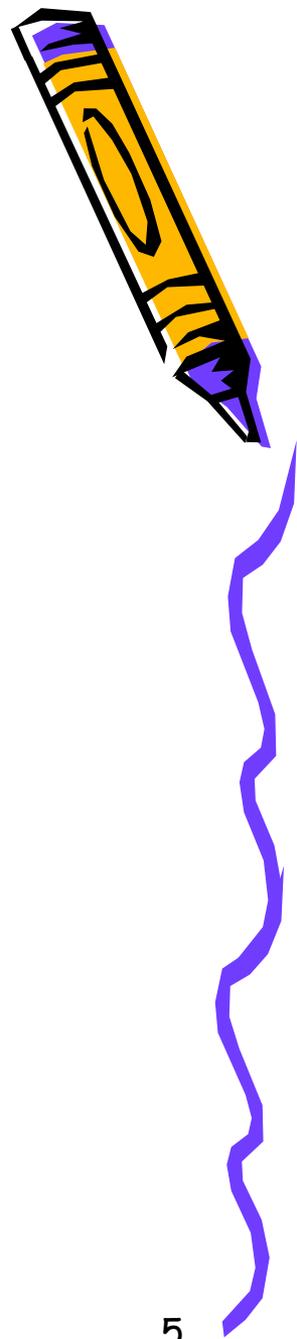
Conditional statements syntax

- The condition is contained within brackets
- It returns a *true* or *false* Boolean value
- Only the statement(s) within the `{ }` will be executed if the condition is true

```
if (1>2) {  
    alert("oops there is something wrong");  
}
```

`{ }` is not a must but better to use, this line of code is also valid:

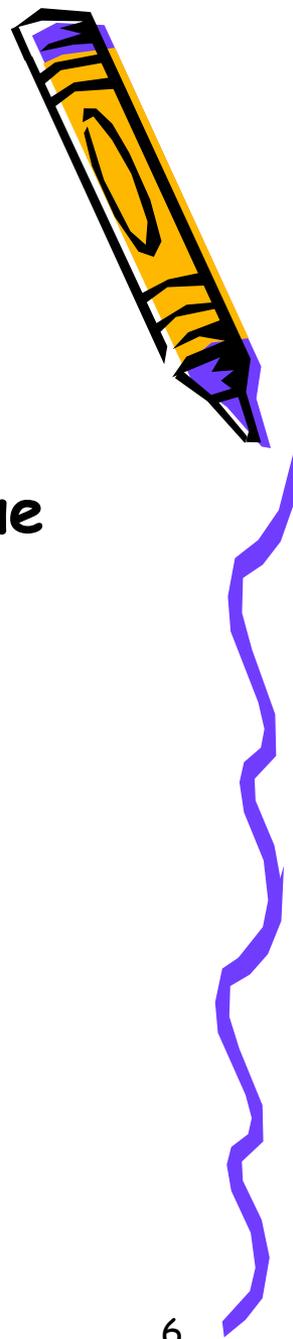
```
if (1>2) alert("oops there is something wrong");
```



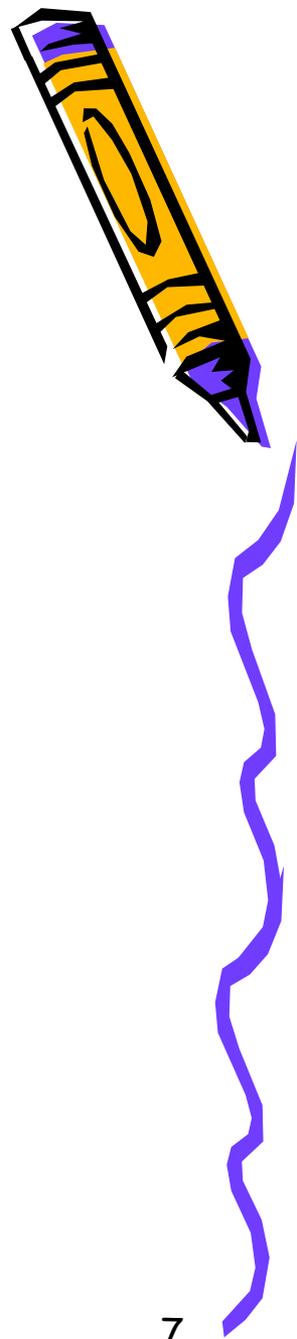
if statement

- Can be extended using *else*
- Statements within the *else* clause will be executed if the condition returns a *false* value

```
if (1>2) {  
    alert("this cannot be right!");  
}  
else {  
    alert("this is correct!");  
}
```



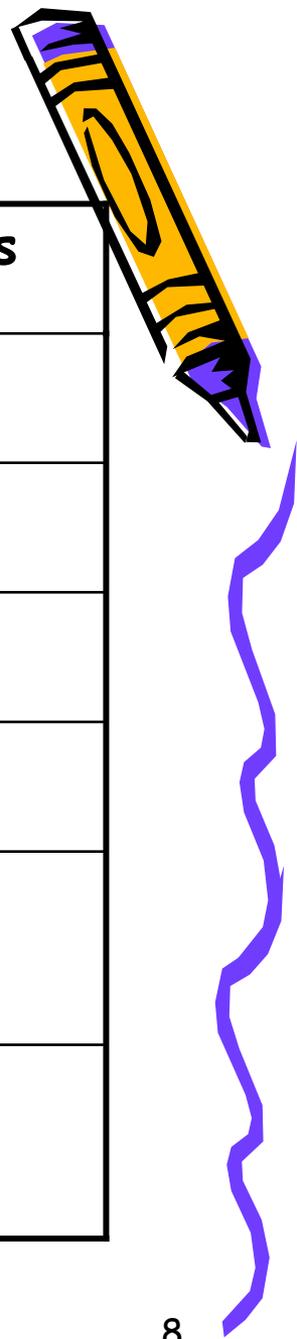
Example Lab3-1



Write an *if* and *else* statement - see lab 3-1 and lab3-1-alert.



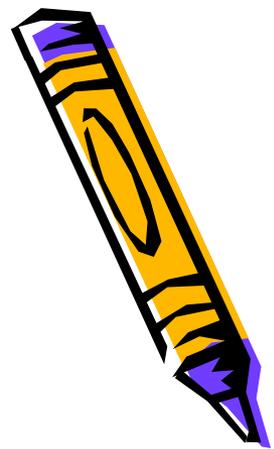
Comparison operators



Operator	Meaning	Syntax	Returns
==	Equals to	$X==y$	false
!=	Not equal to	$X!=y$	true
<	Less than	$X<y$	true
>	Greater than	$X>y$	false
<=	Less than or equal to	$X<=y$	true
>=	Greater than or equal to	$X>=y$	false



example



```
var my_order = "large";  
var your_order = "medium";  
if (my_order = your_order) {  
  document.write("our orders are the same");  
}
```

Wrong way to check for equality

```
var my_order = "large";  
var your_order = "medium";  
if (my_order == your_order) {  
  document.write("our orders are the same");  
}
```

This is the correct way to check for equality

or

```
if (my_order != your_order) {  
  document.write("we have different orders");  
}
```



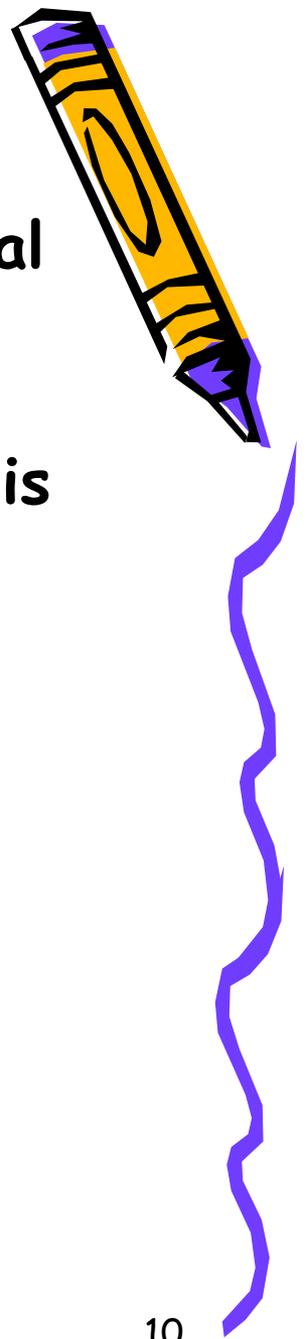
Logical operators (operands)

- Are used to combine conditions in a conditional statement
- For example if I want to check if a variable is not equal to something and is equal to something else:

```
if (order_size>0 && size==small) {  
    alert(" We have orders for size small");  
}
```

Or

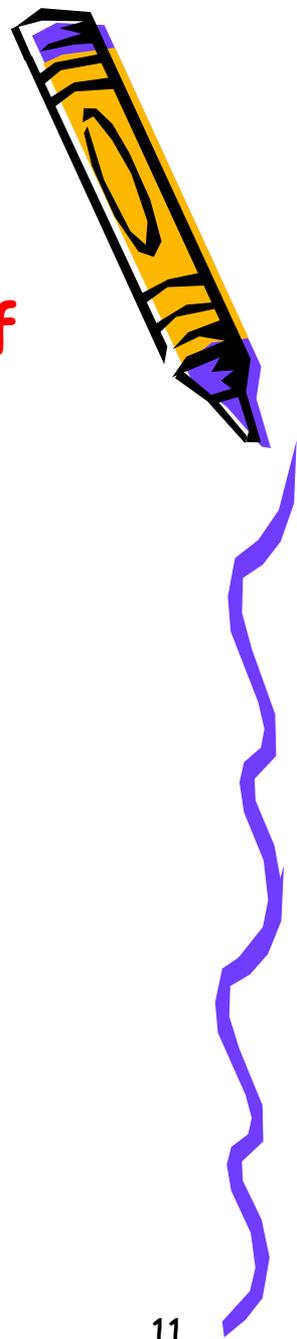
```
if (customer_num>0 || order_num !=0) {  
    alert("We have got an order");  
}
```



Looping statements

Looping statements allow you to repeat specific operations for a specified number of times

- *while*
- *do ... while*
- *for*



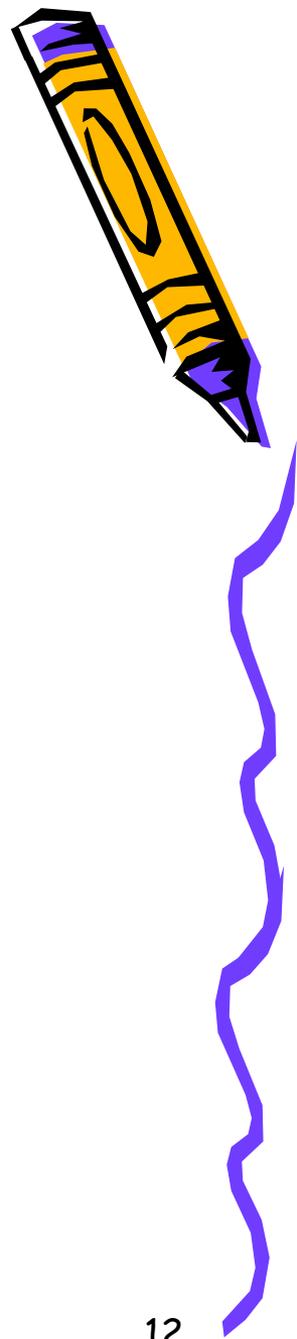
while loop

Similar to if statement:

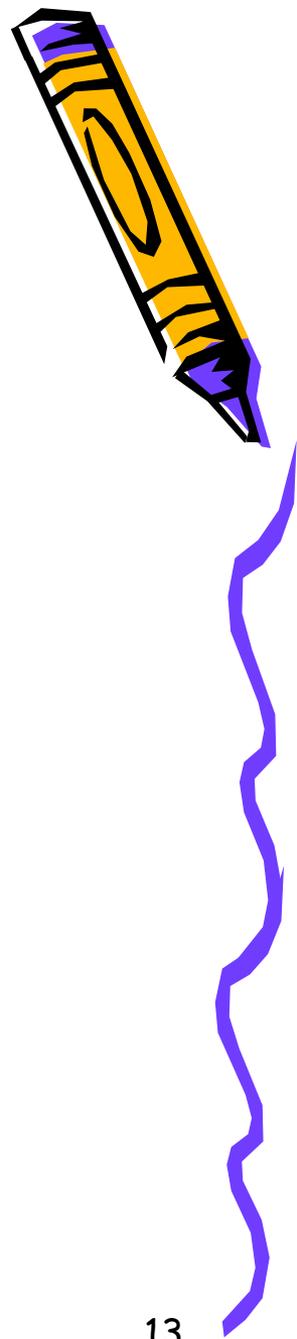
```
while (condition) {  
    statements;  
}
```

The statements will be repeated as long and the condition is true:

```
var count = 0;  
while (count <= 10) {  
    alert (count);  
    count++;  
}
```



Example Lab3-2



Write a *while* loop counting from 1 to 6

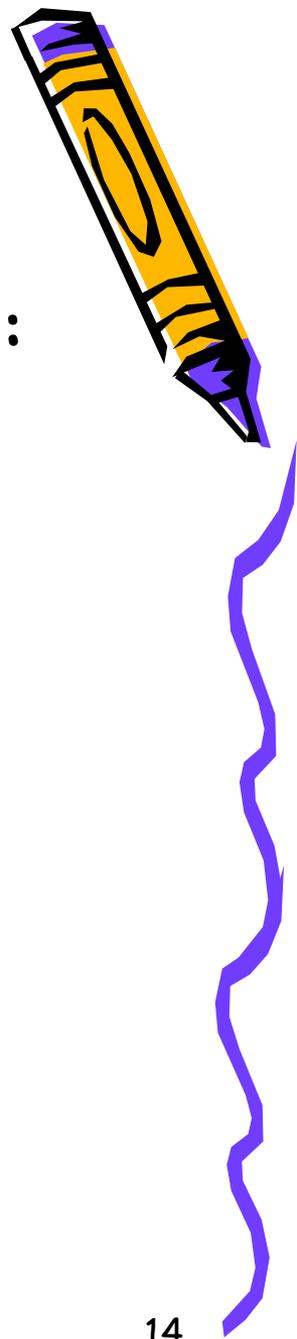


do ... while loop

If you intend to execute a condition for at least once, then you use the *do ... while* loop:

```
do {  
    statements;  
} while (condition);
```

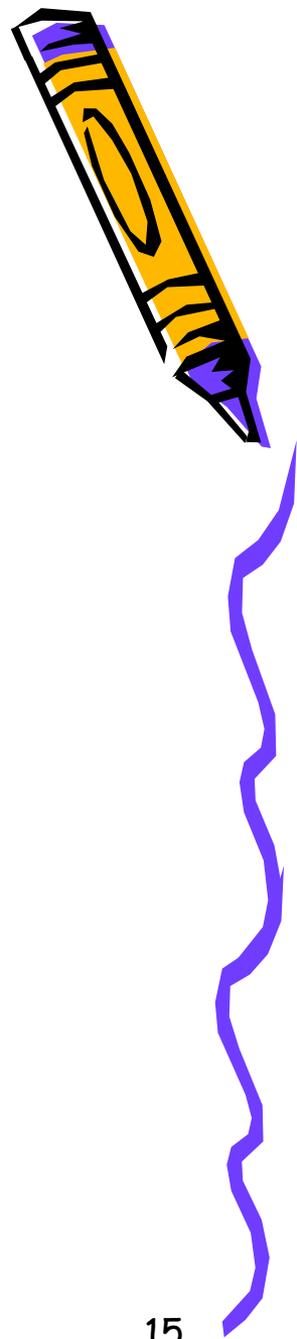
// in this case even if the condition evaluates as false on the first loop, the statements contained within the { ... } will still be executed once.



Compare the two... (lab3-3)

```
var count = 1;  
do {  
    document.write("the number is:" count);  
    document.write("<br />");  
    count++;  
} while (count<=5);
```

```
with  
do{  
    document.write("the number is:" count);  
    document.write("<br />");  
    count++;  
} while (count<1);
```



for loop

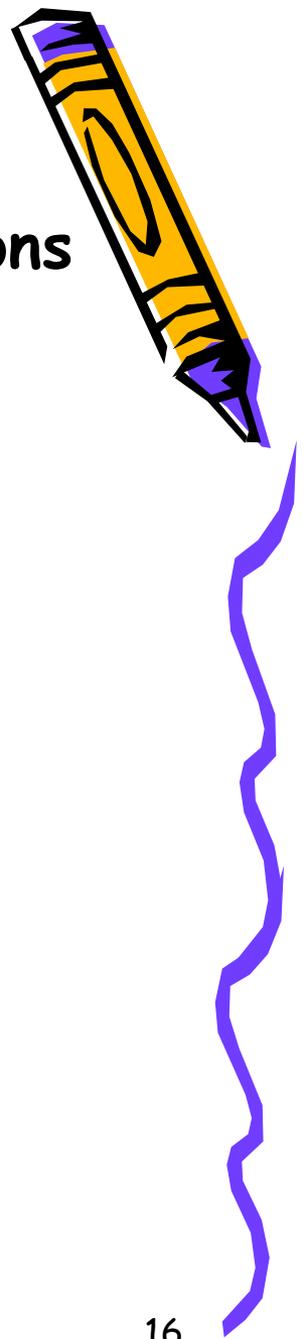
Is a convenient way to repeat a set of operations for a specified number of time:

```
for (initial condition; test condition, alter condition) {  
    statements;  
}
```

```
for (var i = 1, i <= 5; i++){  
    alert (i);  
}
```

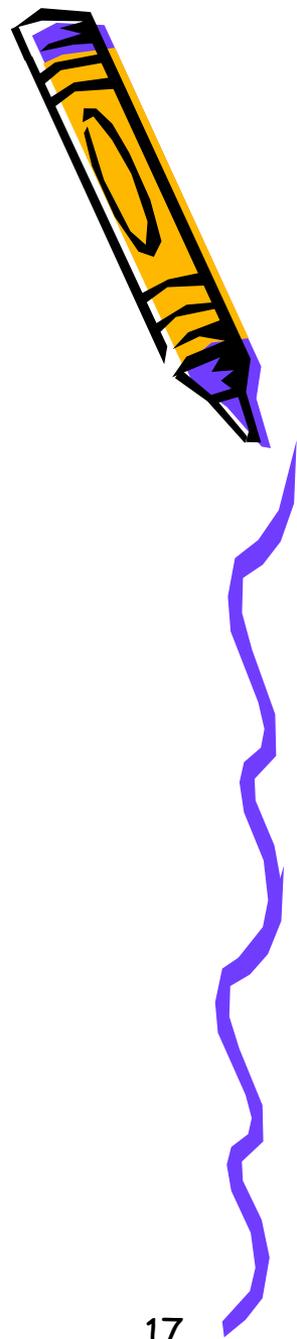
Operations on array elements:

```
var team = Array("me", "you", "her", "him");  
for (var i = 0; i < team.length; i++){  
    alert(team[i]);  
}
```



Example lab3-4

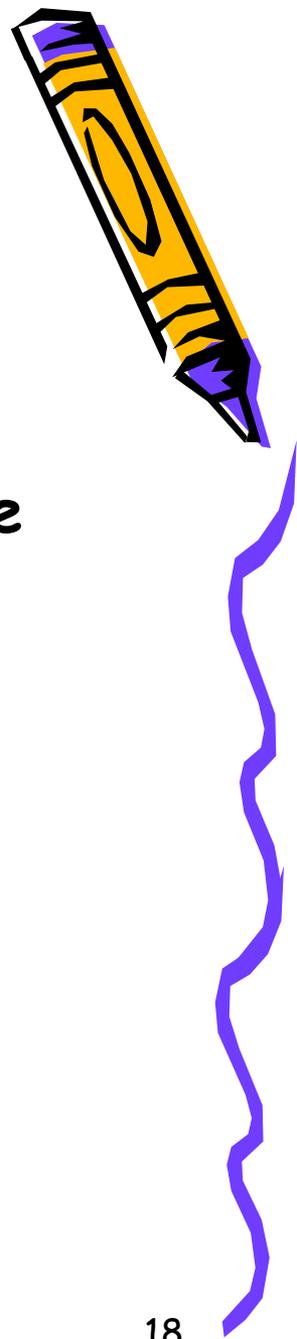
for loop and Array



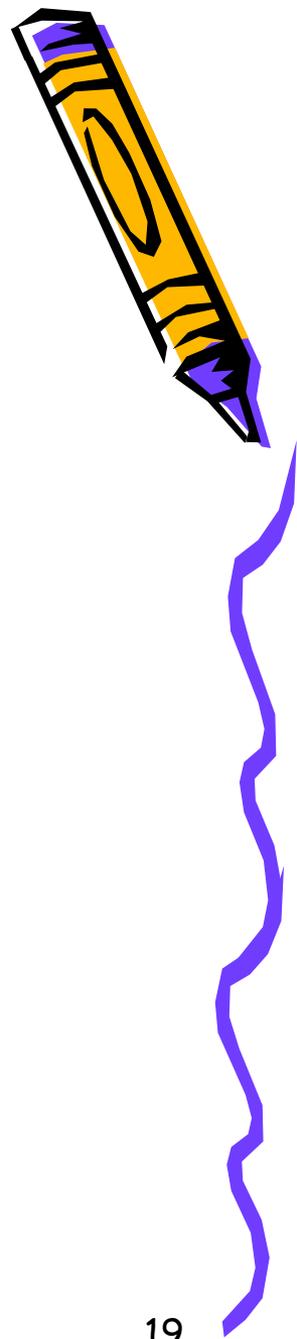
Functions

- For the purpose of reusability of a piece of code you can wrap statements into Functions
- A function is a set of statements that can be invoked from anywhere in your code
- Define your function before invoking it

```
function name(argument){  
    statements;  
}
```



Example lab 3-5

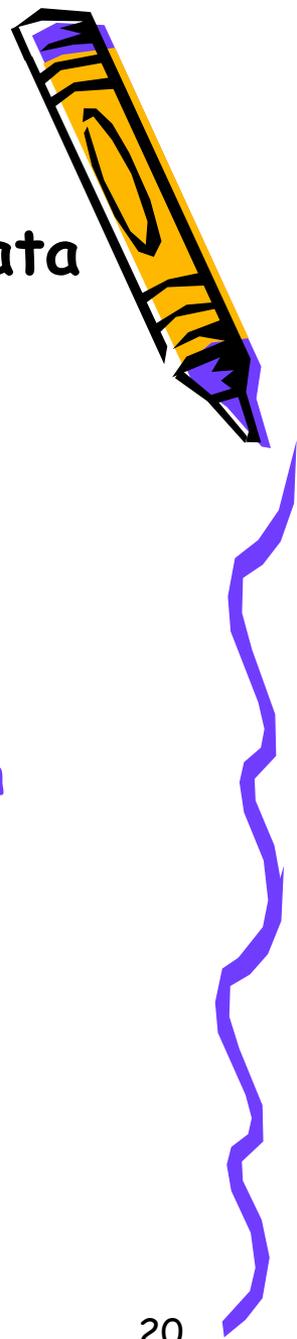


Function to display a message



Objects

- An object is a self-contained selection of data
- It's a neat way to represent data
- Objects consist of properties and methods
 1. **Property** is an attribute (i.e. variable) belonging to an object
 2. **Method** is a function that an object can invoke



Accessing properties and methods

Object.property

Object.method()

For example:

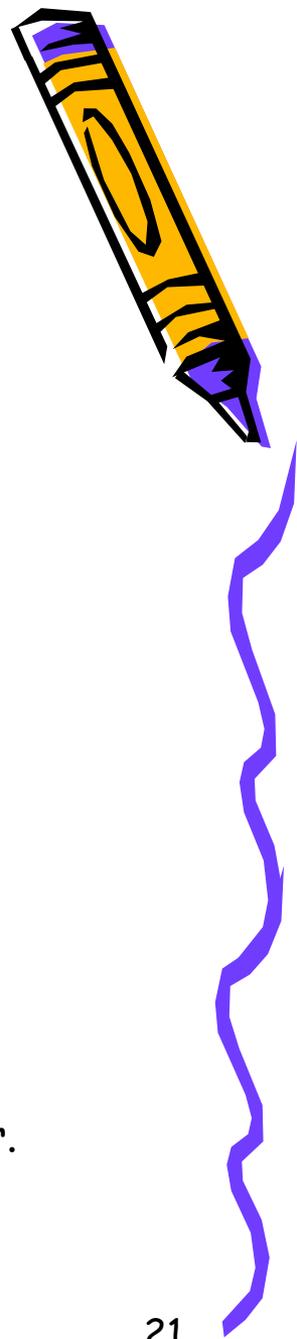
`car.colour`

`car.enginesize`

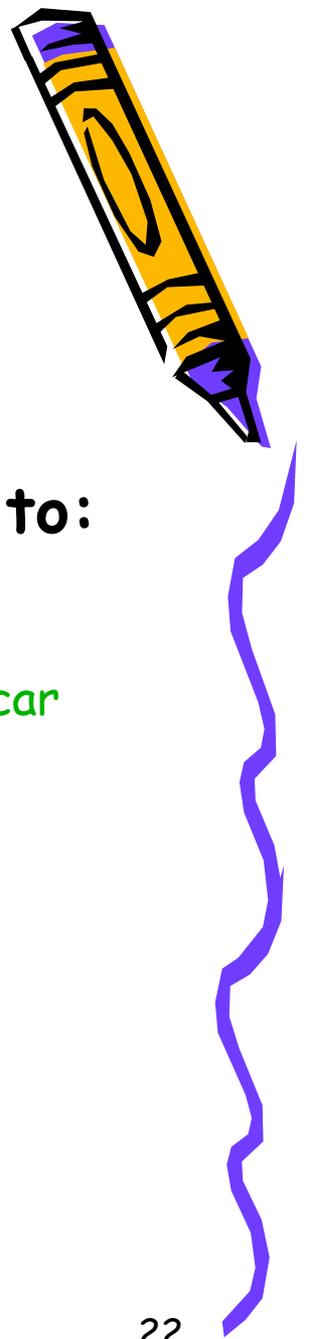
`car.calculatespeed()`

`car.calculatehp()`

All these properties and methods are grouped together under the term `car`.



Instantiating Objects



- You will now use the car Object to describe instances of a specific car specifications
- To create an instance of an object you need to:

```
var toyota = new car; // creates a new instance of the object car  
//called toyota
```

```
toyota.colour
```

```
toyota.enginesize
```



Types of Objects

- User-defined
- Native objects for example `Array`, `Maths.round(num)` `today.getDay()`...
- Host Objects - not part of JavaScript but part of the host application that it runs in, such as: `Form`, `Image` and `Elements`.

