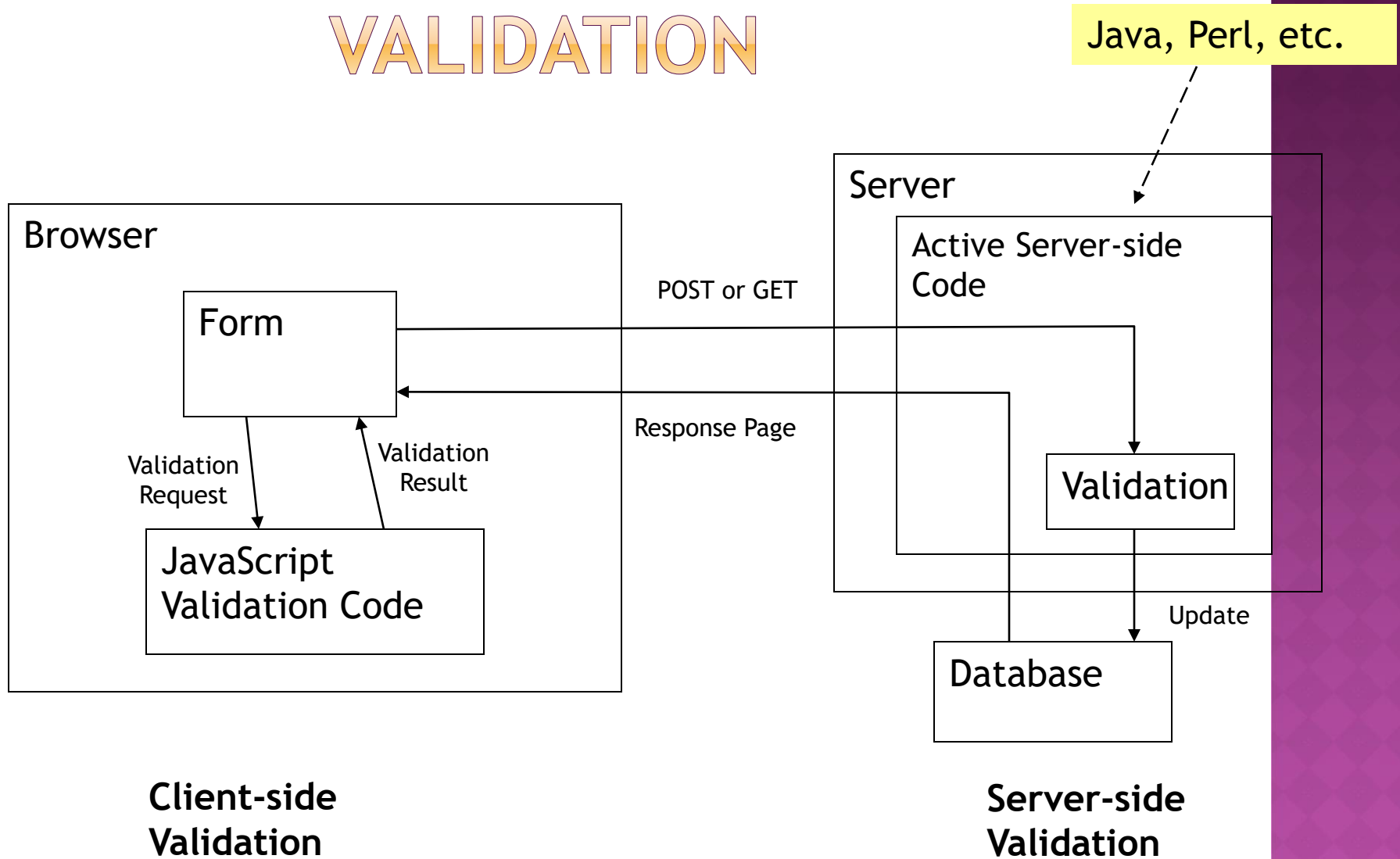


# INPUT VALIDATION WITH JAVASCRIPT

- Client-side user input validation
- Use of JavaScript for form user input validation

Regular Expressions  
Event Handlers

# USER INPUT VALIDATION



# CLIENT-SIDE USER INPUT VALIDATION

- ◉ Avoids round-trip request to server when form has obvious errors
  - Delays (when network or server is slow)
  - Disruption of page transition
  - Unnecessary server traffic
- ◉ Notifying user of error - alternatives:
  - Pop-up “alert box” (annoying, disruptive)
  - Disable submit button plus on-form message (less obvious)

# SERVER-SIDE USER INPUT VALIDATION

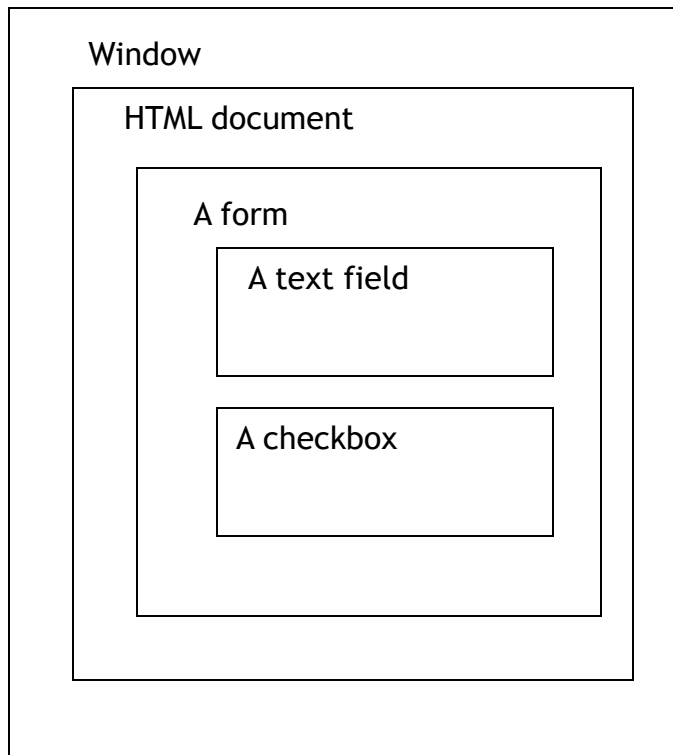
- Client-side validation does not eliminate the need to validate user input on the server
  - A malicious user could copy and modify your page to eliminate the client-side validation
  - Server-side re-validation is crucial if bad user data could be harmful or insecure
  - Client-side validation is to be considered only a convenience for the user and a means to reduce unnecessary traffic on the server
- Sometimes validation is too hard to do on the client

# JAVASCRIPT

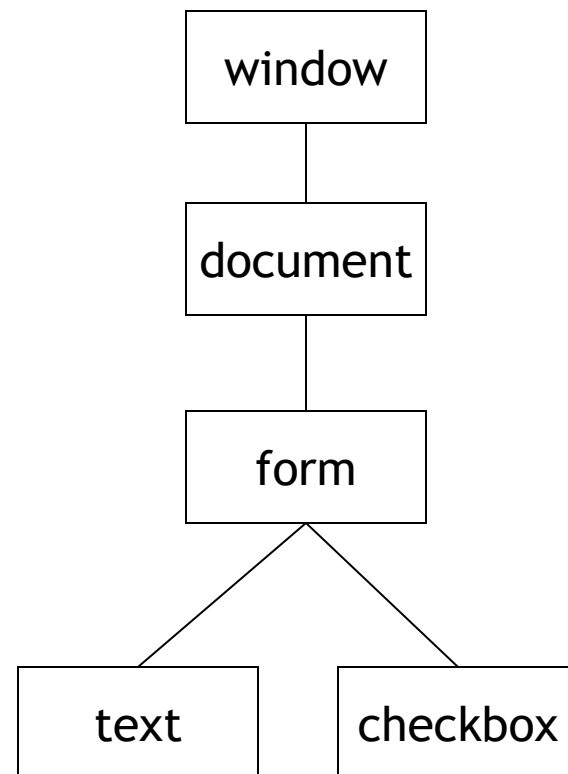
- ◉ Scripting language designed for use in *client-side* programming in web pages
- ◉ In fact, a powerful, full-featured, object-oriented language
- ◉ Originally created by Netscape, which initially called it LiveScript
- ◉ Standardized by the European Computer Manufacturers Association.

# OBJECTS IN A PAGE

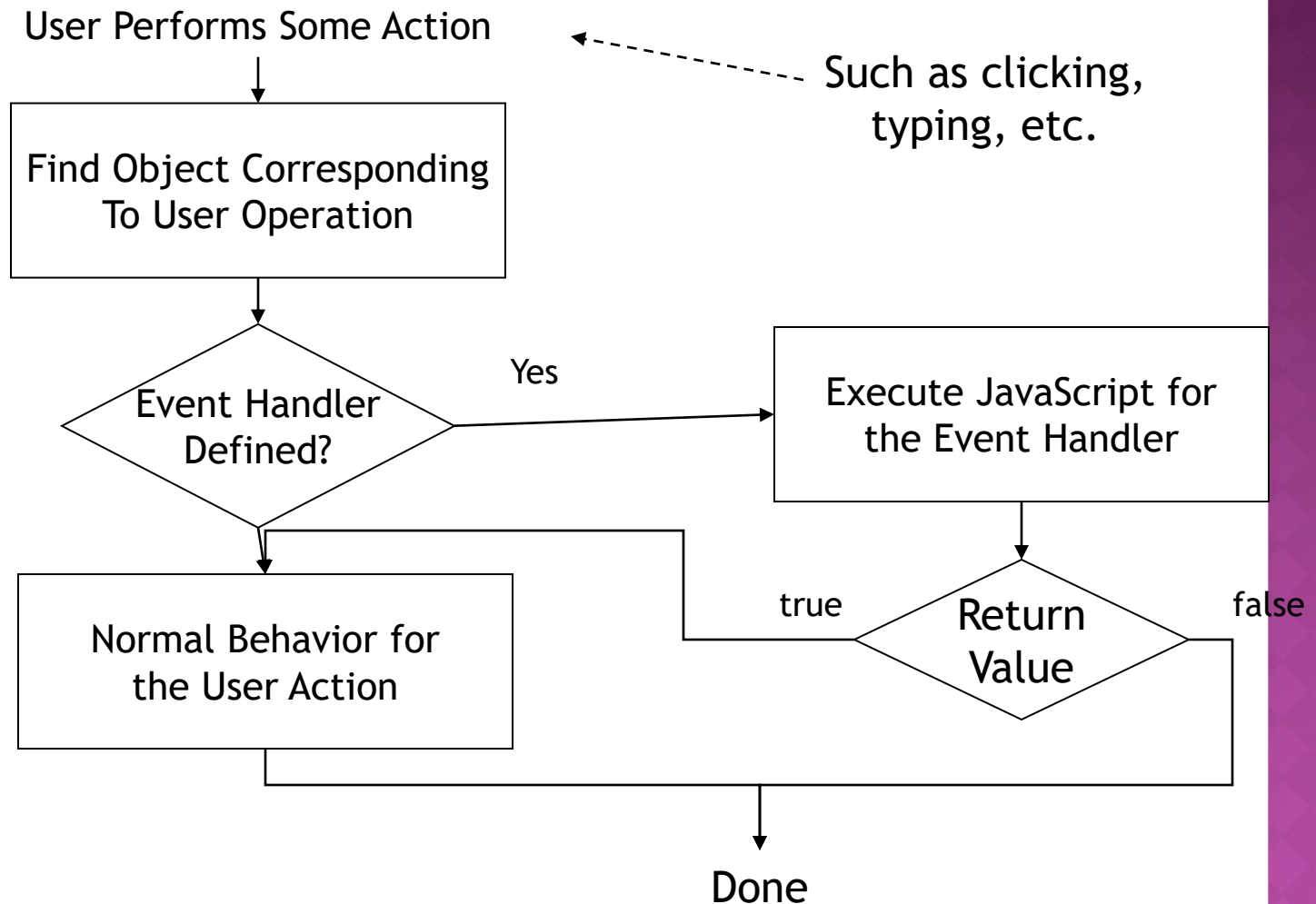
As appearing on the  
screen



As appearing in the JavaScript  
*Document Object Model*



# THE BROWSER INVOKING JAVASCRIPT CODE



# SETTING AN EVENT HANDLER

```
<form id="myForm" method="get" action="..."  
  onsubmit="return check(this)" >
```

- ◉ Declares that the code `return check(this)` will be executed when the form is about to be submitted
- ◉ The code `check(this)` is a call to a function called `check`.
  - The argument *this* refers to the JavaScript object associated with the event (which is the form in this case)
- ◉ The submit will actually occur only if the `check` function returns `true`.



# HOW DO WE DEFINE THE FUNCTION CHECK?

```
<script language="JavaScript">
<!--
function check(form)
{

if(form.t1.value == "")
{
    alert("The text field is empty.");
    return false;
}

if(! form.c1.checked)
{
    alert("The checkbox is not checked.");
    return false;
}
return true;
}
//-->
</script>
```

○ Put this code in the <head>

- It will also work in the <body>, but in the <head> is preferred.

○ The <*script*> tag “hides” the text inside it from HTML parsing

- So <, etc. can be used without problem
- The comment lines are used to prevent problems in older browsers that don't recognize the <script> tag

○ Alternatively, the JavaScript code can be read in from a file - discussed later.

○ What does this code mean?

# DEFINING A FUNCTION

```
<script language="JavaScript">
```

```
<!--
```

```
function check(form)
```

```
{
```

```
  var txtFld = form.t1;
```

```
  if(txtFld.value == "")
```

```
  {
```

```
    alert("The text field is empty.");
```

```
    return false;
```

```
  }
```

```
  if(! form.c1.checked)
```

```
  {
```

```
    alert("The checkbox is not checked.");
```

```
    return false;
```

```
  }
```

```
  return true;
```

```
}
```

```
//-->
```

```
</script>
```

Defines a function and one parameter

- Recall: The parameter is a reference to the form object
- No type is declared for the argument
- No return value type is declared

Declares a local variable called "txtFld" and initializes it

- No type is declared for the variable
- The **var** is optional, but serves to make the variable local, thus preventing collisions with variables called "txtFld" that might be used elsewhere
- **form** contains properties that references all the elements of the form by ID or name

The txtFld var is included here only to illustrate the var concept. It's unnecessary and doesn't appear on subsequent slides.

# DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "")
    {
        alert("The text field is empty.");
        return false;
    }

    if(! form.c1.checked)
    {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

**form.t1.value** refers to the value of the text field named **t1**.

- This tests whether the text field's value is the empty string.
- The attributes of a tag will appear as properties of the object associated with the object.
- Note that *string comparison* is done with the `==` operator (unlike Java)

# DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{
    if(form.t1.value == "")
    {
        alert("The text field is empty.");
        return false;
    }

    if(! form.c1.checked)
    {
        alert("The checkbox is not checked.");
        return false;
    }
    return true;
}
//-->
</script>
```

The **alert** function is built in

- The **alert** function pops up a confirmer box with the given text and an OK button.
- This is an example to illustrate the coding technique. In good design practice, a more detailed, user-friendly message might be given.

The **check** function returns **false**.

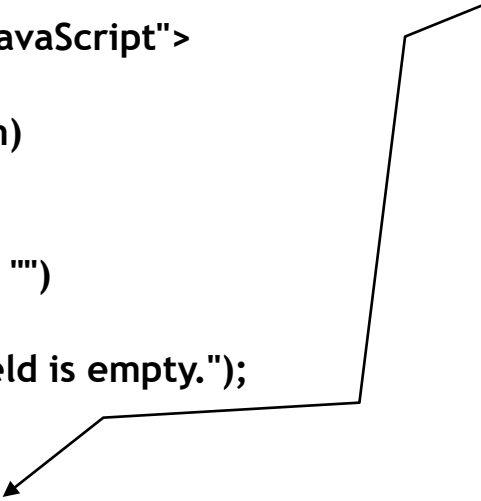
- This tells the browser to **not continue with the submit**.

# DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{

if(form.t1.value == "")
{
    alert("The text field is empty.");
    return false;
}

if( ! form.c1.checked )
{
    alert("The checkbox is not checked.");
    return false;
}
return true;
}
//-->
</script>
```



This tests if the checkbox is checked or not.

- The **checked** attribute is a Boolean.
- The **!** is the NOT operator.
- It is, of course, pointless to have a single checkbox that you require to be checked .
  - This is only an example.
  - Normally, there would be multiple checkboxes and you would verify that at least one of them is checked - or whatever you wish.

# DEFINING A FUNCTION

```
<script language="JavaScript">
<!--
function check(form)
{

if(form.t1.value == "")
{
    alert("The text field is empty.");
    return false;
}

if( ! form.c1.checked )
{
    alert("The checkbox is not checked.");
    return false;
}
return true;
}
//-->
</script>
```

Again there is a popup alert box and the function returns false to indicate that the submit should not occur.

The check function returns **true** if everything is OK. This causes the submit to actually occur.

# HTML FOR A SUBMIT EXAMPLE

```
<html>
<head>
  <title>Submit Example</title>
  <script language="JavaScript">
```

JavaScript from  
previous slides  
goes here.

```
    </script>
  </head>
  <body>
    <form id="myForm" method="get"
      action="javascript:alert('submitted')"
      onsubmit="return check(this);" >
      <input type="text" name="t1" >
      <input type="checkbox" name="c1" value="c1" >
      <input type="submit" >
    </form>

  </body>
</html>
```

Temporary  
action URL for  
testing

# THE JAVASCRIPT URL SCHEME

Notice the use of single quotes inside double quotes

`action="javascript:alert('submitted')"`

- ⦿ The URL uses `javascript` instead of `http` for the scheme.
- ⦿ This is understood by the browser to mean
  - Don't send a `get` request to the server
  - Instead, execute the given text as JavaScript code
  - The resulting value is used as the HTML for a new page
  - To stay on the same page, suppress the value with:  
`void(<expression>)`
- ⦿ This technique can be very useful for debugging and testing - and sometimes useful for production as well.



# WHAT HAVE WE DONE SO FAR?

- ◉ Added an `onsubmit` attribute to the `<form>` tag that calls the check function, passing a reference to the form by using the `this` keyword.
- ◉ Defined the `check` function in a `<script>` tag
  - It tests the values of form fields
  - On error, it pops up an alert box and returns `false`
  - If all is OK, it returns `true`.

# ALTERNATIVE FOR INCLUDING JAVASCRIPT CODE

```
<script language="JavaScript"  
  src="submitExample.js" >
```

```
</script> ← Still need the end tag
```

## ○ Benefits

- Can share JavaScript source among many pages
- Removes a lot of clutter from the page - improves readability
  - Becomes really important with servlet- and JSP-generated pages!
- Helps to separate page design and functionality
- Hides your JavaScript code from the *casual* observer
  - But, of course, one can always access the .js file separately.
  - There are techniques for encrypting the JavaScript file, but we won't go into them.

# GETTING THE VALUES OF FORM FIELDS

Type of <code>&lt;input&gt;</code>	Attribute	Attribute Values
checkbox or radio	<code>checked</code>	true or false
	<code>value</code>	the value to be submitted if true
file	<code>value</code>	the URL of the file to be uploaded
hidden	<code>value</code>	the value to be submitted
text or password	<code>value</code>	the text to be submitted
button, submit, reset	N/A	N/A
<code>&lt;select&gt;</code>	<code>selectedIndex</code>	0-based number of the <code>&lt;option&gt;</code> that is selected
	<code>options[index].value</code>	the value to be submitted
<code>&lt;textarea&gt;</code>	<code>value</code>	the text to be submitted

# SPECIAL CODE FOR ACCESSING RADIO BUTTONS

- Recall, the radio buttons that work as a group are given *the same name*
- Thus, JavaScript uses an **array** to represent them:

```
var radioGroup = form.userRating;  
var rb;  
var valueChecked = "";  
for(rb = 0; rb < radioGroup.length; rb++)  
    if(radioGroup[rb].checked)  
        valueChecked = radioGroup[rb].value;  
alert("The value checked is: " + valueChecked);  
if(valueChecked == "bad") #unfair validation!  
    return false;  
return true;  
}
```

An **array** of all the elements with the name  
userRating

Arrays have a  
built-in **length**  
property.

# VALIDATION UPON FIELD CHANGE

- ◉ Check validation as soon as user changes a field
- ◉ Benefit: more convenient for user to know of error right away
- ◉ Problems:
  - Can't check for required fields
  - Won't prevent submit, so you have to re-validate with onsubmit
  - Can give a spurious error when one of two *dependent fields* are changed
  - Can be annoying if done clumsily

# VALIDATION UPON FIELD CHANGE, CONT.

*onchange* event occurs

- ◉ field is losing focus
- ◉ but only if its value has changed

```
<input type="checkbox" name="opt1"  
      onchange="validate(this)" >
```

```
<textarea cols="30" rows="10" name="comment"  
      onchange="validate(this)" >
```

# THE ONCHANGE EVENT

- ◉ Event occurs on a form *field*.
  - The `this` variable refers to the field, not the form
- ◉ You have the choice of
  - Defining a separate validation function for each field or
  - Using a common validation function that applies different rules for different fields:  

```
if(field.name == "password") ...
```
- ◉ Event fires when the form element loses focus *but only if* its value has changed
- ◉ Gives you the ability to tell users of errors immediately

# EXAMPLE: VERIFY A 5 DIGIT ZIP CODE

```
<input type="text" name="zipcode"
      onChange="validate5Zip(this)" >
```

```
function validate5Zip(field)
{ //verify it's a 5-digit zip code
  var zip = field.value;
  if(zip.length != 5 || isNaN(parseInt(zip)))
  {
    alert("please enter a 5 digit zip code.");
    return false;
  }
  return true;
}
```

Note: Use the keyboard double quote character. These open and close quotes are an artifact of PowerPoint

Converts string to integer and checks if result is "not a number"



# EXAMPLE RECAST AS A SINGLE VALIDATION FUNCTION

- ◉ The validate function:

```
function validate(field)
{
    var val = field.value;
    switch(field.name)
    {
        case "zipcode":
            if(val.length != 5 || isNaN(parseInt(val)))
            {
                alert("please enter a 5 digit zip code.");
                return false;
            }
            break;
        case ...    //Other fields tested here
    }
    return true;
}
```

**<input type="text"  
name="zipcode"  
onchange="validate(this)" >**

**Note the  
switch  
statement  
works with  
strings!**

# ADDITIONAL TOPICS

# ONSUBMIT VALIDATION COMBINED WITH ONCHANGE VALIDATION

```
function check(form)
{
    if(!validateZip(form.zip59) )
        return false;

    if(!form.c1.checked)
    {
        alert("The checkbox is ...");
        return false;
    }
    return true;
}
```

Check for  
required fields

Re-validate fields with  
onchange validation in case  
they remain unchanged from a  
default value:

- Invoke the onchange validation function used in the onsubmit handler
- Avoid duplicated code.

## Sidebar

Be careful of alerts and other messages when you invoke a validation function from inside another.

Avoid duplicate warnings and other annoying behavior.

# CROSS-FIELD VALIDATION

When one field's validation depends on the value of another field:

- ◉ In onsubmit validation, there's no new problem since the `form` object is passed as a parameter
- ◉ In onchange validation, you access the `form` object through the field's `form` property:

```
function validateShipAddress(field)
{
    var isSameAddress = field.form.sameAddressCheck.checked;
    if(!isSameAddress) //if user hasn't said ship address is same
        if(!field.value)
        {
            alert("Please enter a shipping address or check the box"
                + "\"Shipping Address Is Same As Billing Address\"");
            return false;
        }
    return true;
}
```

