

IT614

HTML5 FORMS

1. URL encoding	4
2. Regular expressions	5
➤ The meta-character . (dot)	5
➤ The meta-character *	5
➤ The meta-character +	6
➤ The meta-character ?	6
➤ The meta-character	7
➤ The meta-character ^	7
➤ The meta-character \$	7
➤ The meta-character \	8
➤ The meta-brackets ()	8
➤ The meta-brackets []	9
➤ The meta-brackets { }	10
➤ Special escape strings	11
➤ Some special examples	11
3. HTML Forms	16
4. The input tag	18
➤ The <input> tag with type="submit"	18
➤ The <input> tag with type="text"	19
• The placeholder attribute in input text boxes	20
• The autofocus attribute in input text boxes	20
• The autocomplete attribute in input text boxes	21
• The required attribute in input text boxes	22
• The pattern attribute in input text boxes	22
➤ The <input> tag with type="radio"	23
➤ The <input> tag with type="checkbox"	25
➤ The <input> tag with type="hidden"	26
➤ The <input> tag with type="color"	26
➤ The <input> tag with type="date"	27

➤ The <input> tag with type="datetime-local" _____	27
➤ The <input> tag with type="month" _____	28
➤ The <input> tag with type="week" _____	29
➤ The <input> tag with type="time" _____	29
➤ The <input> tag with type="email" _____	30
➤ The <input> tag with type="tel" _____	30
➤ The <input> tag with type="url" _____	30
➤ The <input> tag with type="number" _____	31
➤ The <input> tag with type="range" _____	32
➤ The <input> tag with type="reset" _____	32
➤ The <input> tag with type="password" _____	32
5. Pull down menu list _____	33
6. Text area boxes _____	34
7. The MAILTO feature _____	35
8. The <datalist> tag _____	35
9. The <keygen> tag _____	36
10. The <output> tag _____	37
11. The <form> attribute onsubmit _____	38
12. The <meter> tag _____	38
13. The <button> tag _____	40
14. More on URL-encoding _____	40
15. Validation of form data _____	41
16. HTML Tags verifying input _____	42
➤ <input type="URL"> _____	42
➤ <input type="email"> _____	42
➤ The pattern attribute _____	42
➤ The min attribute _____	43
➤ The max attribute _____	43
➤ The required attribute _____	43

➤	The maxlength attribute _____	44
17.	<i>Using CSS to display wrong input</i> _____	44
➤	The CSS selectors input : valid and input : invalid _____	44
➤	The CSS selector input : required _____	45
18.	<i>Using CSS to make form output nice</i> _____	45
19.	<i>Customizing error messages</i> _____	48

HTML forms are a very powerful tool for interacting with users. Practically every Web site uses forms to receive input from users – all kinds of information: email addresses, name and address, credit card info and so on. An HTML Form is used to create of one or more widgets. Those widgets can be text fields (single line or multiline), select boxes, buttons, checkboxes, and radio buttons and submit buttons.

The main purpose of an HTML form is to send data input by the user to a server. The server side program will process the data, saves the information in a database, and then send a response to the user. In this course we will not deal with server side programs.

Before we start looking at forms, we will discuss two important topics: **URL-encoding** and **regular expressions**.

Both URL-encoding and regular expressions are related to forms. But regular expressions are a standard tool in many systems and scripting languages.

1. URL encoding

When you fill a form on your browser (such as your name and address, credit card numbers, and so on), and click a submit button, the data you typed will be sent to a program on a server for processing. The data you send will not be sent as is, it will be encoded (I don't mean encrypted) and sent to the server using a URL. Such encoding is called URL-encoding, also called percent-encoding.

The browser uses the following formula in URL-encoding an input data:

A space in input is replaced by a +

The following input characters: +, !, \$, &, ', (,), ,, /, :, ;, =, ?, @, [, and] are replaced by %xx, where xx is the ASCII value for the character in hexadecimal format. In other words, the following table shows the conversion of these special characters:

!	#	\$	&	'	()	+	,	/	:	;	=	?	@	[]
%21	%23	%24	%26	%27	%28	%29	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

All other input characters remain the same unchanged

Examples:

Input string	URL-encoded string
John Smith	John+Smith
It's Raining!	It%27s+Raining%21
My tel # is (212) 773-3702	My+tel+%23+is+%28212%29+773-3702
The answer = 72	The+answer+%3D+72
Jack & Jill went up the hill!	Jack+%26+Jill+went+up+the+hill%21
You can use these: [3, 4, 9]	You+can+use+these%3A+%5B3%2C+4%2C+9%5D
My email is nmurthy@pace.edu	My+email+is+nmurthy%40pace.edu
The cost is \$35.76	The+cost+is+%2435.76
He is 6' tall	He+is+6%27+tall
3 + 2 is 5	3+%2B+2+is+5
15/3 is = 5	15%2F3+is+%3D+5
How much does it cost?	How+much+does+it+cost%3F

Read this again and make sure you understand how the input data is converted into one string without spaces.

2. Regular expressions

Regular expressions appear in many places in IT area. It is important to learn regular expressions well once.

A regular expression is a compact way of describing a set of character strings. A regular expression, also called a pattern, itself is a character string which contains regular characters and characters which represent special meaning. When a character string satisfies a given regular expression, we say that the character string matches the regular expression.

The following characters have special meaning in a regular expression:

. \ | () [] { } ^ \$ * + ?

These characters are also called meta-characters.

A regular expression always matches the longest string. (Explanation on this later.)

A regular expression matches left to right. (Explanation on this later.)

The first time you are exposed to regular expressions, they look very difficult. That is because they are difficult to understand the first time. So study well.

➤ The meta-character . (dot)

A dot (.) in a regular expression has a special meaning: a . in a regular expression matches ANY ONE character.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
a.b	Matches any string with a, followed by ANY ONE character, followed by b.	axb ayb a7b a.b a*b

➤ The meta-character *

A * in a regular expression has a special meaning: a * in a regular expression matches the preceding character zero or more number of times. Read this again. * does NOT match any character string, it matches the preceding character zero or more number of times.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
ab*	Matches any string with a , followed by b zero or more number of times.	abbbbbbbb abb

		a abbb abbbb
--	--	--------------------

Note:

The regular expression **.*** matches any character string.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
a.*b	Matches any string with a , followed by ANY character string with any number of characters, followed by b .	aquwinb ab a19snhb abbb abbbb

➤ **The meta-character +**

A **+** in a regular expression has a special meaning: a **+** in a regular expression matches the preceding character one or more number of times.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
ab+	Matches any string with a , followed by b one or more number of times.	abbbbbbbb abb ab abbb abbbb

Note

ab* matches a, but ab+ does not match a.

➤ **The meta-character ?**

A **?** in a regular expression has a special meaning: a **?** in a regular expression matches the preceding character zero or one time.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
--------------------	---------	---

ab?	Matches any string with a , followed by b zero or <u>one</u> time.	ab a
------------	--	---------

➤ The meta-character |

A | in a regular expression has a special meaning: a | in a regular expression matches either the expression on the left of | or the right of |.

The meta-character | is called an "alternation" character which works like "or".

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
abc xyz	Matches abc or xyz	abc xyz
a.b xy?	Matches regular expression a.b or the regular expression xy?	apb x xy a4b

Note: On a keyboard the | character is in the key



➤ The meta-character ^

A ^ in a regular expression has a special meaning only when it appears as the first character in the regular expression: a regular expression starting with a ^ followed by a character matches a string which starts with that character. In other words, the regular expression ^x matches any string which starts with the character x.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
^a	Matches any string which <u>starts</u> with a	abc a87x

➤ The meta-character \$

A \$ in a regular expression has a special meaning only when it appears as the last character in the regular expression: a regular expression ending with a \$ matches a string which ends with that character preceding the \$ in the regular expression. In other words the regular expression x\$ matches any string which ends with the character x.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
a\$	Matches any string which ends with a	bca 87xa

➤ **The meta-character **

A \ in a regular expression has a special meaning: a \ in a regular expression suppresses the special meaning of the character following it. This is called "quoting." \ is also called escape character.

As you know, these characters ., *, ?, |, ^, and \$ have special meaning in a regular expression. Now, if you want to include these characters in a regular expression, without any special meaning, you must quote them. That is, place a \ before these special characters.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
a\.b	Matches a followed by a dot, followed by b.	a.b
a*b	Matches a followed by a *, followed by b.	a*b
a\ b	Matches a followed by a , followed by b.	a b
a\^b	Matches a followed by a ^, followed by b.	a^b
a\\$b	Matches a followed by a \$, followed by b.	a\$b

Note

We will use \ to quote other special characters, which we will see later.

➤ **The meta-brackets ()**

The meta-brackets (...) are used in a regular expression to group patterns.

Examples:

Regular expression	Meaning	Examples of strings matching the regular expression
(bc nm)d	Matches strings bcd and nmd	bbcd nmd
(a b c d)xyz	Matches a or b or c or d, followed by xyz.	axyz bxyz cxyz dxyz

➤ The meta-brackets []

The meta-brackets are used to define a class of characters.

The set of characters specified within the brackets is said to form a class.

The notation [...] represents any ONE character specified within brackets. Read it again:

Each [] represents ONE character, any one character from the characters within [...].

The special meta-characters lose special meaning within [...].

Different ways of specifying a class:

class	Example	Matches
A class can include individual characters	[QWsd93]	Matches ONE character: Q W s d 9 3
A class can be a range of characters, separated by -.	[d-p]	Any one character from d through p.
	[3-9]	Any one character (digit) from 3 through 9
	[A-Z]	Any uppercase letter
A class can be a multiple ranges and individual characters.	[2-8PQR]	Any one of the characters: 2 through 8 P Q R
	[RBU1-5]	Any one of the characters: R B U 1 through 5
	[0-9a-z]	Any one of the characters:

		0 through 9 a through z
A class with the <u>first</u> character ^ negates the list: [^list] matches any one character other than the characters in the list.	[^BHUp ^r s]	Any one character other than B, H, U, p, r s.
	[^2-6]	Any one character other than 2 or 4 or 5 or 6.

Note

Presence of ^ in a position other than the first has no special meaning. The ^ in this [...^...] has no special meaning.

Examples:

Regular expression	Examples of matching strings
a[0-9]p	a0p a9p a7p a5p
[0-5][s-z]	Two characters (remember each [] represents ONE character): 0s st 4z 5u
[0-9][0-9][0-9]	A three digit number
[A-Z][0-9][a-z]	Three characters: Any capital letter, followed by any digit, followed by a lowercase letter.
[382][d-h]	Two characters: 3d 8e 2g 8h

Important note:

In the class, the range can only be range of characters – NOT a range of numbers. That is, you cannot have [120-345]. This DOES NOT represent a range of numbers from 120 through 345.

➤ The meta-brackets {}

{n} Matches repetition of the preceding character exactly n times.

{n,} Matches repetition of the preceding character at least n times.

$\{n,m\}$ matches repetition of the preceding character at least n times but not more than m times.

Examples:

The regular expression $a\{2\}$ matches aa .

The regular expression $a\{2,\}$ matches aa , aaa , $aaaa$,

The regular expression $a\{2,4\}$ matches aa , aaa , $aaaa$.

➤ Special escape strings

As we mentioned before, `'\'` is called an escape character. The following escape sequences in a regular expressions have special meaning:

Escape sequence	Meaning
<code>\d</code>	Matches a digit. Same as <code>[0-9]</code>
<code>\D</code>	Matches any non-digit character. Same as <code>[^0-9]</code>
<code>\w</code>	Matches any alphanumeric character including the underscore. Same as <code>[A-Za-z0-9_]</code> .
<code>\W</code>	Matches any character other than alphanumeric character including the underscore. Same as <code>[^A-Za-z0-9_]</code> .
<code>\b</code>	Match a word boundary

➤ Some special examples

Special example 1:

Consider the string: $(a+b) * 2 (x+y-z) + (c+d) * (2x+y-3z) * (x+y-4z) + (a+b+c) + a+b$

Now consider the regular expression: `\(.*\)`

This regular expression is matching, a `(`, followed by any string, followed by a `)`. It look like there are several segments in the given string which match the regular expression, as shown below:

$(a+b) * 2 (x+y-z) + (c+d) * (2x+y-3z) * (x+y-4z) + (a+b+c) + a+b$

Which one is the match?

Answer: a regular expression always matches the longest string. That is, the regular expression `\(.*\)` matches everything from the first `(` to the last `)`. Thus the match is really,

$(a+b) * 2 (x+y-z) + (c+d) * (2x+y-3z) * (x+y-4z) + (a+b+c) + a+b$

Here is a challenging question for you: Write a regular expression which matches the shortest string (...). That is the regular expression must match $(a+b)$.

Special example 2:

A regular expression to match IP addresses.

IP address: As you know, an IP address consists of four numbers, each of which contains one to three digits, with a single dot (.) separating each number or set of digits. Each of the four numbers can range from 0 to 255. Here's an example of an IP address: 78.125.168.1.

We need write a regular expression to match IP addresses. Let us do this step by step. Each of the four numbers is between 0 and 255. Let us first write a regular expression to match a number between 0 and 255. As you know we cannot have a range of [0-250].

So let us split the problem of representing 0 through 255 into:

One digit numbers.

Two digits numbers.

Three digit numbers between 100 – 199.

Three digit numbers between 200 – 255.

Let us discuss regular expressions for the first three of these:

For this group	Regular expression
One digit numbers	\d
Two digits numbers	\d\d
Three digit numbers 100 – 199	1\d\d

We can combine these three to one regular expression: [01]?\d\d?

Now a regular expression for a three digit number between 200 - 255.

Let us split this problem to these two steps: 200 – 249 and 250-255.

A regular expression for numbers 200-249: 2[0-4]\d.

A regular expression for 250-255: 25[0-5].

Combining all these ideas, a regular expression for a number between 0 and 255 is: [0-1]?\d\d?|2[0-4]\d|25[0-5].

Now, we need a dot at the end of these (first three groups of the IP):

([0-1]?\d\d?|2[0-4]\d|25[0-5])\.

We need three groups of this: (([0-1]?\d\d?|2[0-4]\d|25[0-5])\.){3}

The last group does not have a dot at the end: [0-1]?\d\d?|2[0-4]\d|25[0-5]

Thus a regular expression for an IP address:

(([0-1]?\d\d?|2[0-4]\d|25[0-5])\.){3}([0-1]?\d\d?|2[0-4]\d|25[0-5])

Special example 3:

A regular expression to match simple email addresses.

Assume that an email address:

Starts with a character string consisting of letters, numbers, points, underscores, plus and minus

Followed by @

Followed by a string consisting of letters, numbers, points, underscores

Followed by a period

Followed by a string consisting of 2 to 4 letters

Regular expressions for these 5 items:

[a-zA-Z0-9._+-]+

@

[a-zA-Z0-9._]+

\.

[a-zA-Z]{2,4}

Thus, a regular expression for a simple email address is

[a-zA-Z0-9._+-]+@[a-zA-Z0-9._]+\.[a-zA-Z]{2,4}

Special example 3:

A regular expression to match social security numbers.

A simple version: **\d{3}-\d{2}-\d{4}**

Special example 4:

A regular expression to match 10-digit telephone numbers in the format (201) 645-2928.
Homework.

Practice Examples:

1.

Which of the strings below does the regular expression [2468][0-9][0-9] match?	
432 567	Match
34	No match
14567	Match. matches 456 in 14567
sam236	Match. Matches 236 in sam236
45Bill9	No match
999	No match

2.

Which of the strings below does the regular expression \b[2468][0-9][0-9]\b match?	
432 567	Match
34	No match
14567	No Match.
sam236	No Match.
45Bill9	No match
999	No match

3.

Which of the strings below does the regular expression [1-386] match?	
1	Match
57	No match
4	No match
5	No match
49	No match
3	Match

4.

Which of the strings below does the regular expression [2-4]{2}[a-d]{2} match?	
24db	Match
22aa	Match
42ab	Match
44cd	Match
123abc	Match. Matches 23ab in 123abc

5.

Which of the strings below does the regular expression [^abc]+[abc]+ match?	
abc	No match
pppppaa	Match
xabc	Match
xyz	No match
4xc5pa	Match.
567bca	Match

6.

Which of the strings below does the regular expression a+b*c+ match?	
abc	Match
bc	No Match
abbbbcccc	Match
abbbbbbb	No match
aaabbbbcccc	Match.
a	No Match
b	No Match
c	No Match
aacccccccc	Match

7.

Which of the strings below does the regular expression [0-9][0-9]? dollars? match?	
14 dollars	Match
1 dollar	Match
234 dollars	Match. Matches 34 dollars in 234 dollars
save 10 dollars	Match. Matches 10 dollars in save 10 dollars

0 dollar	Match.
1234 dollars	Match. Matches 34 dollars in 1234 dollars

8.

Which of the strings below does the regular expression [1234]\.-+\w+ match?	
1.---4	Match
2.-Tiger	Match
3.--man	Match
44.--bird	No Match
3.snake	No Match
5.-----dog	No Match
1234---cat	No Match

3. HTML Forms

Let us start the feature presentation now: HTML forms. Form elements and attributes have been there in all versions of HTML. But, in HTML5, forms provide a greater degree of semantic mark-up than earlier versions.

A form is a feature by which you ask the reader of the page for input and process the data on the server. The form tag is **<form>...</form>**.

Typical syntax:

```
<form>  
    Actual form elements go here  
</form>
```

Three of the commonly used **<form>** attributes are **name**, **method** and **action**.

The name attribute

name="formName"

Assigns a name to the form. This name is used by programs which process the data entered into the form by the user.

The method attribute

Commonly used values for attribute **method** are **get** and **post**. This value decides how the browser communicates the data with the server program.

The action attribute

The value of the attribute **action** is a URL to a program on the server.

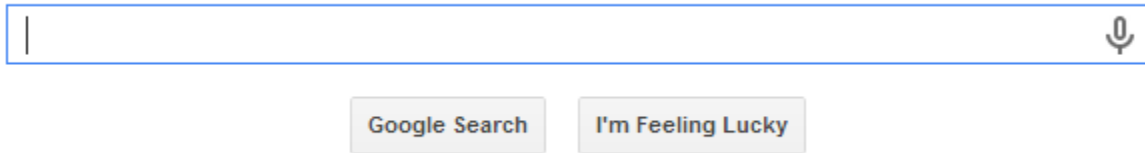
The **<form>** tag contains several **<input>** tags, which actually receive data from the user. When the user inputs data in the input tags and clicks the submit button, the data is sent to the program on a server specified in the **action** attribute for processing of the data.

Differences between **method="get"** and **method="post"**

When the user types data in form (you will see how to do it below when we discuss **<input>** tags) and clicks the submit button, the data typed by the user is sent to a program on a server whose URL is indicated as the value of the **action** attribute in the **<form>** tag.

When you use **method="get"**, the data typed by the user is concatenated into one URL-encoded string and sent to the server side program. This string is available to the program on the server in an environment variable **QUERY_STRING**. When you click the submit button of the form, you can actually see the URL-encoded string being sent to the server in the address bar of the browser. See the following example to understand this point:

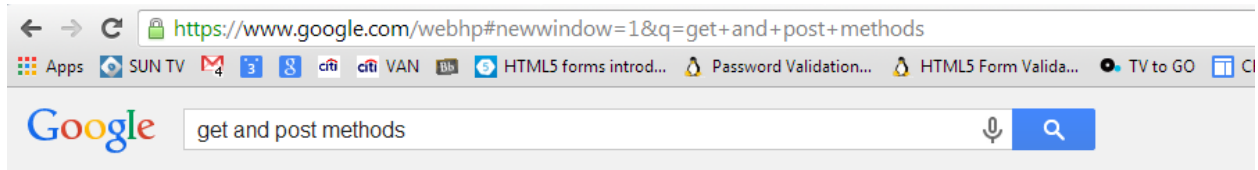
Start Google search page.



```
href="https://plus.google.com/u/0/me/about?wXvView_profile/<a></div><div>div class=""gb_2 gb_gb">a class=""</div>  
href="https://www.google.com/webhp?authuser=0"><img class=""gb_6" src=""//lh6.ggphtusercontent.com/-xHtU_OWhqAg/photo.jpg" alt="Narayan Murthy"><div class=""gb_7"><div class=""gb_8">Narayan Murthy</div><div class=""gb_9">profnurmurthy@gmail.com (default)</div><div>  
<a></div>a class=""gb_aa gb_gb" href="https://plus.google.com/u/0/dashboard"><span class=""gb_ba gb_g"></span>All your Google+ pages &rsquo;<a></div>  
class=""gb_X"><div>a class=""gb_D" href="https://accounts.google.com/AddSession?hl=en&page=continue&https://www.google.com/webhp#Add_account"></div>  
<div>a class=""gb_jb gb_gb gb_D" href="https://accounts.google.com/Login?hl=en&page=continue&https://www.google.com/webhp#target=top">Sign  
out</div><div></div><div></div><div><div class=""gb_g gb_gb j">id="gbq1" style="max-width:127px;min-width:127px"><div class=""gb_ha">a  
class=""gb_kb gb_ja" href="//webhp?tab=w&page=ic3eqGUseLzChESAT37HwQd&page=vedoCBQBI3s" title="Go to Google Home"><span class=""gb_g gb_g"></span><a></div>  
</div><div class=""gb_g gb_g gb_Wa"><div id="gbq"><div class=""gb_gbqf">id="gbqg2"><div id="gbqgf"><form class=""gb_pb" action="/search" onsubmit="" target=""  
id="gbqf" name="gbqf" method="get"><fieldset class=""gbqx"><legend class=""gbqx">Hidden fields</legend><div id="gbqgfd"><input name="newwindow" value="1"  
type="hidden"><input name="output" value="search" type="hidden"><input name="scient" value="psy-b" type="hidden"></div><fieldset><fieldset  
class=""gbqff gb_g" id="gbqgf"><legend class=""gbqx"></legend><div id="gbqfa class=""gbqfa "><div id="gbqfg class=""gbqfg"><div id="gbqfgb class=""gbqfgw>  
<input id="gbqfg class=""gbqfig name=q type=text autocomplete=off value=""></div></div></div></div><div><div class=""gb_g gb_gb"><button  
class=""gbqfg" aria-label="Google Search" name="btnG" id="gbqbf" class=""gbfl gb_g"></span></button></div><div class=""jsb">id="gbqfbwa"><button
```

Back on the search page, type a search string (try get and post methods),





The string `q=get+and+post+methods` is the URL-encoded string of your input. You know what `+` represents in the URL-encoded string.

In contrast to this, in the **Post** method, the user input from the form will be passed to the server side program script as standard input. You don't see the URL-encoded string in the address bar.

Get methods leave the parameters in browser history because they are part of the URL. **Post** methods don't do that.

There is a limitation on the amount of data (about 2K) you can send using get methods. Such restrictions are not there when you use post methods.

Clearly, **get** methods are less secure than post methods.

4. The input tag

There are a variety of input tags. The `<input>` tag is used within **`<form>...</form>`**. The **`<input>`** tag is an empty tag (no end tag). It contains attributes only. There are several kinds of input tags. We will discuss some of them now. Before we discuss data input tags, let me introduce the **submit** button.

➤ The `<input>` tag with `type="submit"`

A submit button is created by using an **`<input>`** tag with **`type`** attribute value **"submit"**.

Syntax

```
<input type="submit" name="buttonName" value="buttonString">
```

The value **submit** of the attribute **type** specifies a clickable button.

The value of the attribute **name** is used to identify the button.

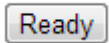
The value of the attribute **value** specifies the caption on the button. If you don't include the value attribute, the default caption is **Submit** in Chrome (Submit Query in IE).

When the user clicks the submit button, the data in the entire form is sent to the program indicated by the URL value of **action** attribute in the form tag.

Example:

```
<input type="submit" value="Ready">
```

Browser output:

**➤ The <input> tag with type="text"**

An input text box is a box in which the user can type data.

Here is an example of how a text box looks in a browser:

Type your name:

The input text box tag takes several attributes as shown in the syntax below.

Syntax:

```
<input type="text" name="textboxName" size=number>
```

The value **text** of the attribute **type** specifies a textbox.

The value of the attribute **name** is used by the server side processing program to retrieve data entered in the textbox.

The value of the attribute **size** specifies the width of the textbox (in number of characters).

The default size is 20.

After completing the text box, the user will click the **submit** button in the form. The data is then sent to the program specified by the **action** attribute in the **<form>** tag.

Practice Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<head><title>A title for the page</title></head>
```

```
<body>
```

```
<form method="get" action="http://vulcan.seidenberg.pace.edu/~nmurthy/cgi-bin/IT614/Example6.pl">
```

```
Type your name: <input type="text" name="name"><P>
```

```
Type your Telephone number: <input type="text" name="telephone"><P>
```

```
<input type="submit" value="Click">
```

```
</form>
```

```
</body>
```

```
</html>
```

Notice that the **action** attribute is pointing to a program:

<http://vulcan.seidenberg.pace.edu/~nmurthy/cgi-bin/IT614/Example6.pl>.

I have actually written a CGI script in Perl and it processes the data. In this simple example, it just replies with a thank you note.

When you open this HTML page in a browser, you will see,

Type your name:

Type your Telephone number:

Enter name (John Smith) and telephone number ((914) 773-3920) and click the button. The data actually goes to the program, Example6.pl and processes the data. In this example, it just replies,

Thank you John Smith. I have noted your telephone number, (914) 773-3920

Try it:

<http://vulcan.seidenberg.pace.edu/~nmurthy/IT614/Chapter6/PracticeExample6.html>

Note: When you try it, notice the URL-encoded string in the browser address bar after you click submit button.

- **The placeholder attribute in input text boxes**

The **placeholder** attribute helps us provide a hint to the user what to input in the text box.

Syntax:

```
<input type="text" name="boxName" placeholder="hint text">
```

Example:

```
<input type="text" name="boxName" placeholder="Type your last name">
```

Output in Chrome:

See the text "type your last name" in light gray. When the user starts typing, this text vanishes.

- **The autofocus attribute in input text boxes**

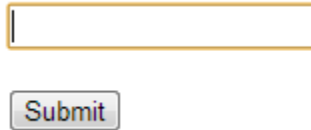
The **autofocus** attribute puts the focus automatically on this text box when the browser loads the form. The **autofocus** attribute does not take a value (it is boolean). If it is present, it will have autofocus.

Syntax:

```
<input type="text" name="boxName" autofocus>
```

Example:

```
<input type="text" name="boxName" autofocus>
```

Output in Chrome:

Notice that the box is "highlighted" and the cursor is in the box.

- **The autocomplete attribute in input text boxes**

The **autocomplete** attribute automatically completes the input, if the box has been completed earlier. Once the user starts typing in the box, depending on the characters the user inputs, it shows a list of previously input data for the user to select one (or type new data). Possible values for the attribute **autocomplete** are "on" or "off."

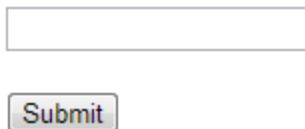
autocomplete="on" is default.

Syntax:

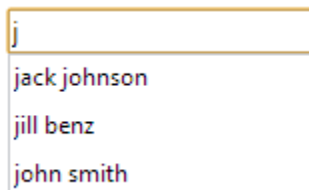
```
<input type="text" name="boxName" autocomplete="on">
```

Example:

```
<input type="text" name="boxName" autocomplete="on">
```

Output in chrome:

For this example, I have filled the box already with data (starting with j). When I type j in the box now,



- **The required attribute in input text boxes**

The **required** attribute forces the user to input data in this box before submitting. The **required** attribute does not take a value (it is boolean). If it is present, data will be required in this box.

Syntax:


```
<input type="text" name="boxName" required>
```

Example:

```
<input type="text" name="boxName" required>
```

Output in Chrome:

See what happens when you try to submit without any data in the box:

  Please fill out this field.

- **The pattern attribute in input text boxes**

The **pattern** attribute forces the user to input data in this box in a particular format. The value of **pattern** attribute is a regular expression. The input data must satisfy the regular expression.

Syntax:

```
<input type="text" name="boxName" pattern="regularExpression">
```

Example:

```
<input type="text" name="boxName" pattern="[A-Z].*">
```

Output in Chrome:

Notice that the regular expression specifies that the input data must start with a capital letter. Notice what happens when you type a name starting with a lowercase letter:

➤ The `<input>` tag with `type="radio"`

Radio buttons are used to provide the user several options and ask the user to make one choice. To make the choice the user clicks a radio button.

Here is an example of how a radio button group looks in a browser:

Give your age group

- ☐ Less than 19
- ☐ Between 20 and 29
- ☐ Between 30 and 59
- ☐ More than 60

The input radio button takes several attributes as shown in the syntax below.

Syntax:

```
<input type="radio" name="radioGroupName" value="buttonValue">
```

The value **radio** of the attribute **type** specifies a radio button.

The value of the attribute **name** is used by the processing program to retrieve data entered in the radio button group. All the radio buttons in a group must have the same value for the attribute **name**.

The value of the attribute **value** is the data sent to the server if this radio button is selected.

There are two more attributes in input radio buttons: **checked** and **disabled**.

If the attribute **checked** is included, the button will be selected by default. The user can, of course, select a different item.

Example:

```
<INPUT TYPE=RADIO NAME=AGE VALUE=2 checked> Between 20 and 29
```

If the attribute **disabled** is included, the button will be grayed and the user will not be able to select it.

Example:

<INPUT TYPE=RADIO NAME=AGE VALUE=4 disabled> More than 60

Example:

Give your age group

- ☐ Less than 19
- ☒ Between 20 and 29
- ☐ Between 30 and 59
- ☐ More than 60

The "Between 20 and 29" is **Checked** and "More than 60" is **disabled**.

Practice Example:

```
<!DOCTYPE html>
<html>
<head><title>Radio Buttons Example</title></head>
<body>
<p>
Give your age group<p>
<form method="get" action="http://vulcan.seidenberg.pace.edu/~nmurthy/cgi-
bin/IT614/Example7.pl" >
<input type="radio" name="age" value=1> less than 19 <p>
<input type="radio" name="age" value=2> between 20 and 29 <p>
<input type="radio" name="age" value=3> between 30 and 59 <p>
<input type="radio" name="age" value=4> more than 60 <p>
<input type="submit" value="OK">
</form>
</body>
</html>
```

Open this in a browser to see:

Give your age group

- ☐ less than 19
- ☐ between 20 and 29
- ☒ between 30 and 59
- ☐ more than 60

OK

When you select a radio button and press OK button, you will see result from the server side program: You are between 30 and 59.

Try It:

<http://vulcan.seidenberg.pace.edu/~nmurthy/IT614/Chapter6/PracticeExample7.html>

➤ The <input> tag with type="checkbox"

Check boxes are used to provide the user several options and ask the user to make as many choices as he/she wants. To make the choices the user clicks the check boxes. In radio buttons you can only select one. In check boxes you can select several.

Here is an example of how an input check box group looks in a browser:

Click all computer programming languages you know:

☐ Python

☐ FORTRAN

☐ C

☐ Java

☐ C++

The input check box takes several attributes as shown in the syntax below.

Syntax:

<input type="checkbox" name="checkBoxName" value="buttonValue">

The value **checkbox** of the attribute **type** specifies a check box button.

The value of the attribute **name** is used by the processing program to retrieve data entered in the check box group.

Practice Example:

```
<!DOCTYPE html>
<html>
<head><title>Check Box Example</title></head>
<body>
<p>Select all computer programming languages you know:<p>
<form method="get" action="http://vulcan.seidenberg.pace.edu/~nmurthy/cgi-
bin/IT614/Example8.pl"><p>
<input type="checkbox" name="pascal"> Pascal <p>
<input type="checkbox" name="fortran"> Fortran <p>
<input type="checkbox" name="c"> C <p>
<input type="checkbox" name="cplusplus"> C++ <p>
```

```
<input type="checkbox" name="java"> Java <p>
<input type="Submit" value="Submit">
</form>
</body>
</html>
```

Browser output:

Select all computer programming languages you know:

☐ Pascal
☐ Fortran
☐ C
☐ C++
☐ Java

Select all the languages and click Submit.

Try It:

<http://vulcan.seidenberg.pace.edu/~nmurthy/IT614/Chapter6/PracticeExample8.html>

➤ The <input> tag with type="hidden"

Hidden fields are similar to text fields, except that the hidden box is not visible on the browser. When the user clicks the submit button, the value in the hidden field is also sent to the server. This is used to send the server certain default values independent of the user.

Syntax:

```
<input type="hidden" name="boxName" value="aValue">
```

➤ The <input> tag with type="color"

The color input type allows the user to select a color and send their selection of color to the server as a hex value.

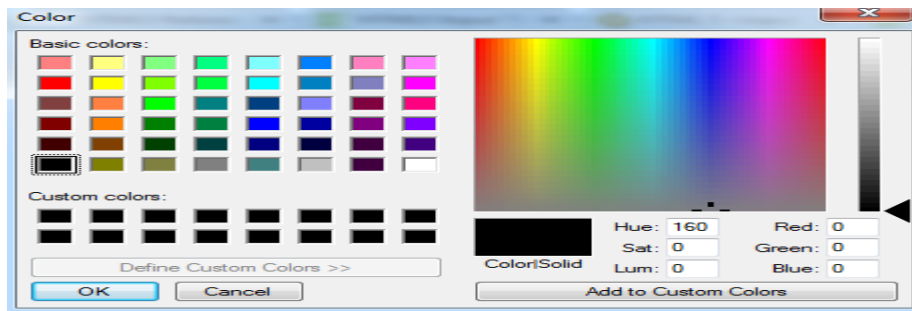
Syntax:

```
<input type="color" name="myColor">
```

Chrome output:



When you click this box, you will see the color picker UI chart:



You can select a color from this chart and click OK.

➤ The `<input>` tag with `type="date"`

This provides a nice interface for the users to input date.

Syntax:

```
<input type="date" name="mydate">
```

Chrome output:

You can select a date by clicking the down arrow button,

December, 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

When you select a date (for example),

And click submit button, the date will be sent to the server in the format, mydate=2013-12-27

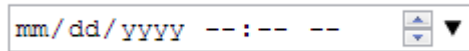
➤ The `<input>` tag with `type="datetime-local"`

This provides a nice interface for the users to input local date and time.

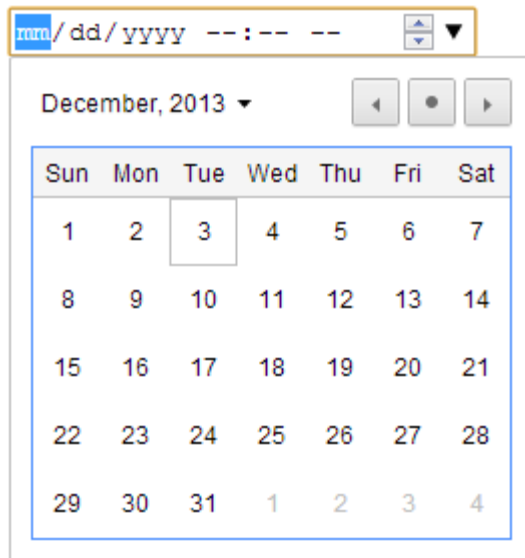
Syntax:

`<input type="datetime-local" name="myDatetimeLocal">`

Chrome output:



You can click the down arrow to select date,



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Once you select a date and input time, and submit, the data will be sent to the server.

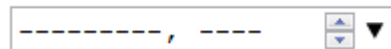
➤ **The <input> tag with type="month"**

This provides a nice interface for the users to input month and year.

Syntax:

`<input type="month" name="myMonth">`

Chrome output:



To input month and year, you can use the down arrow,

December, 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

When you select a month and year, and click submit, the data will be sent to the server.

➤ The `<input>` tag with `type="week"`

This provides a nice interface for the users to input the week number of the year.

Syntax:

```
<input type="week" name="myWeek">
```

Chrome output:

Week 49, ----

To input week number, you can use the down arrow,

Week 49, ----

December, 2013

Week	Sun	Mon	Tue	Wed	Thu	Fri	Sat
49	1	2	3	4	5	6	7
50	8	9	10	11	12	13	14
51	15	16	17	18	19	20	21
52	22	23	24	25	26	27	28
1	29	30	31	1	2	3	4

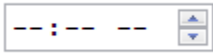
When you select a week number, and click submit, the data will be sent to the server.

➤ The `<input>` tag with `type="time"`

This provides a nice interface for the users to input a time.

Syntax:

```
<input type="time" name="myTime">
```

Chrome output:

When you input a time, and click submit, the data will be sent to the server.

➤ **The <input> tag with type="email"**

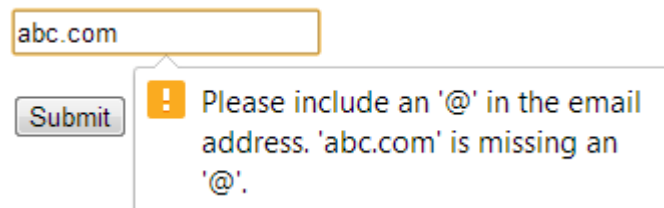
This provides a nice interface for the users to input an email address.

Syntax:

```
<input type="email" name="myEmail">
```

Chrome output:

When you input an email address in the box, and click submit, the data will be sent to the server. If the input does not contain @, it will give an error:



➤ **The <input> tag with type="tel"**

This provides a nice interface for the users to input a telephone number.

Syntax:

```
<input type="tel" name="myTelephone">
```

In Chrome it just gives you a text box. Mistakes in input are currently not identified by Chrome.

➤ **The <input> tag with type="url"**

This provides a nice interface for the users to input a URL.

Syntax:

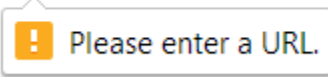
```
<input type="url" name="myUrl">
```

Chrome output:

It just shows a textbox:

When you type a URL, it checks for some pattern, if it is ok, it will be submitted to the server when you click submit button.

If the input does not satisfy certain pattern, it will give an error message:

➤ The <input> tag with type="number"

This provides a nice interface for the users to input a number.

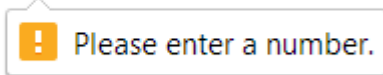
Syntax:

```
<input type="number" name="myNumber">
```

Chrome output:

When you type a number, it will be submitted to the server when you click submit button.

If the input contains non digits, it will give an error message:

You can include more attributes in the <input> tag with type="number":

```
<input type="number" name="myNumber" min=m max=n step=s value=v>
```

The attribute **min** specifies the minimum input permitted, **max** specifies the largest input permitted, **step** specifies the increments when click the up/down arrow, and **value** is the starting value when you load the page.

Example:

```
<input type="number" name="myNumber" min=10 max=100 step=5 value=50>
```

Chrome output:

Because **step** value is 5, every time you click the up arrow, the number jumps by 5 and every time you click the down arrow, the number is reduced by 5. Also, it will not let you enter a number beyond the specified min (10) and max (100).

➤ The <input> tag with type="range"

This provides a nice interface for the users to input a value between certain values using a slider. The user will not know the exact value specified by the slider, but it will give an idea.

Syntax:

```
<input type="range" name="myRange" min=m max=n value=50>
```

The attribute **min** specifies the range minimum, **max** specifies range maximum, and **value** is the starting value when you load the page.

Example:

```
<input type="range" name="myRange" min=10 max=100 step=5 value=50>
```

Chrome output



When you move the slider, and click submit, an appropriate value will be sent to the server.

➤ The <input> tag with type="reset"

A reset button is used to reinitialize all the input values in the form elements to the original value (which may be empty).

A reset button is created using the <input> tag with **type** value "reset".

Syntax:

```
<input type="reset" >
```

The output button, by default contains the word Reset.

We can change the caption of this button by using the attribute VALUE in the tag:

```
<input type="reset" value="Clear">
```

The output button in this case will contain the word clear.

➤ The <input> tag with type="password"

A password box is an input text box in which the user can type data, but the typed data will not be visible in the box (instead you see *****).

A password box is created using the input tag, with **type="password"**.


```

please type your password
<form method=get action="">
<input type=password size=10 name="pwd"><p>
<input type=submit value=ok>
</form>
practice example 8:
<title> password box example</title>
<form method=get action=""><p>
please type your password<p>
<input type=password size=10 name=passwd><p>
<input type=submit value=ok><p>
<input type=reset value=clear>
</form>

```

5. Pull down menu list

A pull down menu is a list of items displayed when a button is clicked. The user can make selections from the list.

A pull down menu is created using the **<select>...</select>** tag within the **<form>** tag. Within a **<select>** tag, we need to include empty **<option>** tag for each item to be listed.

Syntax

```

<select name="aName" size=aNumber>
  <form> an item
  <form> an item
  ...
  ...
</select>

```

The value of the **name** attribute is a name for the select control. The value of **size** attribute specifies how many items to be displayed (visible) in the list.

Note: The **size** attribute does not seem to work correctly in some browsers.

The **<select>** attribute multiple

By default, users can select only one item from the drop down menu. This is like radio buttons control. The **<select>** tag supports another attribute called **multiple**. If **multiple** attribute (no value) is present in **<select>**, then the user can select more than one item from the menu list (by using CTRL key). With the **multiple** attribute, a **<select>** control works like checked boxes control.

The **<option>** tag attributes:

value – The value of this attribute will be sent to the server.

label – The value of this attribute specifies a label for the option.

selected – No value. If **selected** attribute is present, the option will already be selected when the page is loaded (of course, the user can unselect it).

disabled – No value. If **disabled** attribute is present, the option will be grayed out and the user will not be able to select it.

Example:

```
<form method=get action="">
<select name="language" multiple>
<option disabled> Pascal
<option> C
<option> C++
<option selected> Java
<option> Python
</select>
<input type=submit value=ok>
</form>
```

6. Text area boxes

The **textarea** control is rectangular area to type text. A text area is created using the **<textarea>...</textarea>**. The text between the start tag and the end tag will be displayed in the text area, which can be edited by the user.

<textarea> attributes

Attribute	Description
rows = <i>number</i>	Specifies the number of rows visible. You can have more text than what is visible, but you will have a scroll bar.
cols = <i>number</i>	Specifies the visible width of the text area.
name = <i>"aName"</i>	Specifies a name for the text area. Used in sending data to the server.
maxlength = <i>number</i>	Specifies the maximum number of characters permitted in the text area.
disabled	No value. If disabled attribute is present, the text area will be disabled. User will not be able to edit
required	No value. If required attribute is present, the text area cannot be left blank.
wrap = <i>"value"</i>	Value must be either "soft" or "hard" . If the value is soft , no wrapping is done. This is the default behavior. If the value is hard , text area is wrapped. When the value is hard , the attribute cols must be specified.
readonly	No value. If the attribute readonly is present, the text area is read only. The user will not be able to edit it.

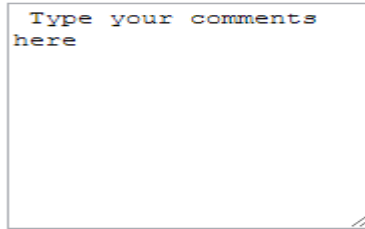
Example:

```
<title> text area example </title>
please send your comments .....
<form method=get action="">
<textarea name=comments rows=10 cols=20 maxlength=10 required>
Type your comments here </textarea>
```

```
<p>
<input type=submit>
</form>
```

Browser output:

please send your comments



Submit

7. The MAILTO feature

The input from a form can be sent as email. In order to accomplish this, just place the e-mail address after **action=** in the **<form>** tag.

Example:

```
<html>
<title>mailto in forms example</title>
<body>
<form method=get action="mailto:actual email address">
<textarea name=comments rows=10 cols=20> </textarea><p>
<input type=submit value=ok><p>
<input type=reset value=clear>
</body>
</html>
```

8. The <datalist> tag

The **<datalist>...</datalist>** is used in conjunction with **<input>** tag. This works like pull-down menu with added feature of "auto completion." This is useful for long lists. Instead of making users scroll through the list, they can use autocomplete feature.

Syntax:

```
<input list="listName">
<datalist id="listName">
  <option> item
  <option> item
  <option> item
  ...
  ...
</datalist>
```

Example:

```
<!DOCTYPE html>
<html>
<head><title>Programming Languages</title></head>
<body>
  <form method="get">
    <input list="languages">
    <datalist id="languages">
      <option> Java
      <option> C++
      <option> C
      <option> JavaScript
      <option> Python
      <option> PHP
      <option> Fortran
      <option> Basic
    </datalist>
    <input type="submit">
  </form>
</body>
</html>
```

Browser output (Chrome):

A screenshot of a web browser interface. It features a text input field containing the letter 'J'. Below the input field, a dropdown menu is visible, showing two options: 'Java' and 'JavaScript'. To the right of the input field is a button labeled 'Submit'.

When you type J, for example, in the box, it lists all the items starting with J.

9. The <keygen> tag

The **<keygen>** tag goes within a **<form>** tag. It is an empty tag (no end tag). This tag is new in HTML5.

This tag generates a private-public key pair, keeps private key on the local machine and send the public key to the server program mentioned in the action attribute of the form tag.

Syntax:

```
<keygen name="aName">
```

The value of the name attribute is used by the server program to retrieve the generated public key.

Other attribute:

keytype

This attribute specifies the algorithm used in key pair generation. Possible values are rsa, dsa, and ec. The default is rsa.

Example:

```
<!DOCTYPE html>
<html>
<head><title>Programming Languages</title></head>
```

```

<body>
  <form method="get" action="http://vulcan.seidenberg.pace.edu/~nmurthy/cgi-
bin/IT614/keyPair.pl">
    Key Pair: <keygen name="keyPair">
      <input type="submit">
    </form>
  </body>
</html>

```

Output:

Key Pair:

This gives you an option of 2048 or 1028. When you click the submit button, the public key is sent to the program specified in the action attribute of the form tag.

For this example, I have written a Perl program just to send the public key back to the browser. When you click the Submit button, it displays the public key (a public key is a long string):

Your public key is,

MIICQDCCASgwgEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCt
0rctWEn4Iq+6LT7KHcPiSmU2WfRJZUvaKOchzo65TTuv8sMBE7hpP/Qh1XKK+o
mGt6SIMb+86GHPH9W8wkTIQ.

Note: If you are not familiar with private-public key encryption, study it on the Web. It is an important concept to know in security area.

10. The <output> tag

The <output>...</output> tag is used to display output of a calculation. Before we discuss <output> tag, a couple of other features.

input type number

type="number" in <input> tag specifies that input is a number. This results in a textbox, with a control to increase or decrease the value. See example below.

Example:

```
<input name="aName" type="number">
```

This produces a textbox as follows:

The valueAsNumber Property

The value input by the user in a "type=number" box can be retrieved using the syntax,

nameOfThebox.valueAsNumber.

Example:

```
<input name="first" type="number">
```

first.valueAsNumber returns the value typed by the user as a number.

11. The <form> attribute onsubmit

In order to use <output>, we need to use **onsubmit** attribute in <form> tag, with the value, "return false." This concept is related to JavaScript. We will not use this in this class.

Syntax for using onsubmit:

```
<form onsubmit="return false"..... >
```

The <form> attribute oninput

In order to use <output>, we need to use oninput attribute in <form> tag. The value for **oninput** is the computation and assigning the result to the <output> box. Details in an example below.

A complete <output> example

```
<!DOCTYPE html>
<html>
<head><title>Using output tag</title></head>
<body>
  <form onsubmit="return false" oninput="result.value = first.valueAsNumber +
second.valueAsNumber">
    <input name="first" type="number"> <P>
    <input name="second" type="number"> <P>
    <output name="result"></output>
  </form>
</body>
</html>
```

Browser output:

first number

second number

first number + second number = 7

12. The <meter> tag

The `<meter>...</meter>` tag is used for indicating a scalar measurement within a known range, or a fractional value. The display is actually a bar graph. The `<meter>` tag was introduced in HTML 5.

Syntax:

`<meter value=number min=0 max=100></meter>`

This is for showing percentage.

Example:

```
<!DOCTYPE html>
<html>
<title>Spoken language</title>
<body>
Percentage of world pollution speaking languages<p>
Mandarin 14.1%<br>
Spanish 5.85%<br>
English 5.52%<br>
Hindi 4.46%<br>
Arabic 4.23%<br>
French 1.12%<p>
Mandarin: <meter value="14.1" min=0 max=100>14.1%</meter><br>
Spanish: <meter value="5.85" min=0 max=100>5.85%</meter><br>
English: <meter value="5.52" min=0 max=100>5.52%</meter><br>
Hindi: <meter value="4.46" min=0 max=100>4.46%</meter><br>
Arabic: <meter value="4.23" min=0 max=100>4.23%</meter><br>
French: <meter value="1.12" min=0 max=100>1.12%</meter><br>
</body>
</html>
```

Output in Chrome:

Percentage of world pollution speaking languages

Mandarin 14.1%

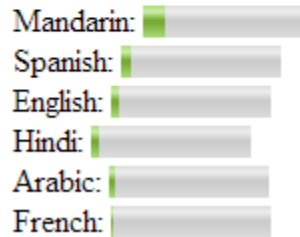
Spanish 5.85%

English 5.52%

Hindi 4.46%

Arabic 4.23%

French 1.12%



13. The <button> tag

We have already used `<input type="submit"...">` to create submit buttons. There is another way of creating buttons: using `<button>` tag.

Syntax:

```
<button type="submit" name="buttonName" > Caption </button>
```

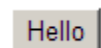
The value of `name` attribute is sent to the server. The text `Caption` will be the caption of the button. In `<input>` buttons, value of the attribute **value** is used as caption.

Also, notice that `<button>` has an ending tag, `</button>`.

Example:

```
<button type="Submit" name="mybutton"> Hello </button>
```

Result:



Differences between `<input type="submit" ...>` and `<button type="submit" ...>`

1. Both produce clickable buttons
2. `<button>` has an ending tag, `</button>`. `<input>` tag does not have an ending tag.
3. The caption in `<button>` is the text between `<button>` and `</button>`. The caption in the `<input>` submit button, is the value of the *value* attribute.
4. `<button>` elements are much easier to style than `<input>` elements.
5. Earlier versions of IE had bugs in submitting the form correctly.

14. More on URL-encoding

You know how the browser converts data input by a user into a URL-encode string. There is one more point related to this.

When you have different pieces of data in a form from different form element, the browser actually does three things: (1) converts data from each element in form to a URL-encoded string and (2) assigns each URL-encoded string to the value of the name attribute in the element and (3) concatenates all these independently formed string into one string delimited by '&'. And this entire string is sent to the server side program.

It is easier to understand with an example,

Consider two input text boxes:

```
<form>
<input type="text" name="one">
<input type="text" name="two">
<input type="submit" value="Send">
</form>
```

This HTML code gives you two text boxes:



Assume you input John Smith in the first box, and 914 77303702 in the second box.

This is what the browser does:

- (1) Creates URL-encode string of each piece of data: John+Smith, and 914+7733702.
- (2) Assigns these URL-encoded string the values of name attributes of boxes. Notice the names of these text boxes are one and two. Thus form the strings, one=John+smith, and two=914+7733702.
- (3) Concatenate these two string by using &: one=John+smith&two=914+7733702.

This string, one=John+smith&two=914+7733702 will be sent to the server.

When you click the send button, you will see this string in the address bar.

15. Validation of form data

The data input by the user needs to be checked for valid data (email should be in the correct format, credit card numbers must have the correct digits and so on). One of way of checking the validity of data is on the server when the data is submitted by the user by clicking the submit button. When the server side program identifies an error in the input, it has to send a message to the user and the user has to input the correct data and submit again. This process wastes network and server resources. The server side validation is still required, but including client-side validation helps sending correct data to the server so that the server when it checks will not find mistakes. And client side checking is quicker than a

round trip to the server, and gives your users instant feedback. This'll keep your users happy, and will have them coming back to your site.

We will now discuss various ways of client-side validation.

In olden days, client-side validation was done using JavaScript. But, HTML5 provides several features to validate data without JavaScripts.

16. HTML Tags verifying input

We have seen most of these earlier.

➤ `<input type="URL">`

The value typed must be a proper URL.

Example:

When you try to submit URL in incorrect format, you will see an error message:

Type a URL

❗ Please enter a URL.

➤ `<input type="email">`

The value typed must be a proper email address.

Example:

When you try to submit email address in incorrect format, you will see an error message:

Type your email

❗ Please include an '@' in the email address. 'jsmith' is missing an '@'.

➤ The pattern attribute

You can use **pattern** attribute with these input types: text, search, url, tel, email, password. The value of a pattern is a regular expression. The value typed by the user must match the regular expression.

Example:

Input one of the New England States: `<input type="text" pattern="CT|ME|MA|NH|RI|VT">`

Input one of the New England States:

! Please match the requested format.

Example:

Your Name (Start with capital letter) `<input type="text" pattern="[A-Z].*">`

Your Name (Start with capital letter)

! Please match the requested format.

➤ **The min attribute**

You can use this attribute in input types number and range.

Example:

Type a number more than 99 `<input type="number" min="100">`

Type a number more than 99

! Value must be greater than or equal to 100.

➤ **The max attribute**

You can use this attribute in input types number and range.

Example:

Type a number less than 1000 `<input type="number" max="999">`

Type a number less than 1000

! Value must be less than or equal to 999.

➤ **The required attribute**

The required attribute can be used with input types text, search, url, tel, email, password, date, datetime, datetime-local, month, week, time, range, number; also on the `<select>` and `<textarea>` elements.

Example:

Type a password `<input type="password" required>`

Type a password

❗ Please fill out this field.

➤ The maxlength attribute

The maxlength attribute can be used with input types text, search, url, tel, email, password; also on the <textarea>element.

Example:

Type message (10 characters max) <textarea rows=3 cols=10 maxlength=10>
</textarea>

Type message (10 characters max)

This constraint will not let you type more than 10 characters.

17. Using CSS to display wrong input

We can use CSS to identify incorrect input by the user.

➤ The CSS selectors **input : valid** and **input : invalid**

These are used with valid/invalid input.

```
input : valid{
```

```
}
```

```
input:invalid{
```

```
}
```

You can provide appropriate CSS declarations in each {...}.

Example:

```
<style>
```

```
input:invalid {
```

```
    border: 5px solid red;
```

```
    background-color:yellow;
```

```
}
```

```
input:valid {
```

```
    border: 1px solid green;
```

```
}
```

```
</style>
```

Now consider the <input> tag:

Your Name (Start with capital letter) <input type="text" pattern="[A-Z].*">

Your Name (Start with capital letter)

The user is expected to type a name in the box starting with a capital letter. See what happens when the user makes a mistake,

Your Name (Start with capital letter)

Note:

We can similarly use selectors:

```
input:required{
```

```
}
```

➤ The CSS selector `input : required`

```
input:invalid {  
  border: 5px solid red;  
  background-color:yellow;  
}
```

```
input:valid {  
  border: 1px solid green;  
}
```

Your Name <input type="text" required>

In the beginning (when the box is blank), it will satisfy "input:invalid". Once you start typing in the box, it will satisfy "input:valid".

Your Name

18. Using CSS to make form output nice

The default look of form widgets is not very good. You have to make them look nice by using CSS style sheets

There are plenty of examples on the Web, which show how to accomplish this. You will be using the usual CSS properties. There are new selectors. I will mention one important new selector:

input [type="typeValue"]

This selects all <input> elements with specified types.

A complete example

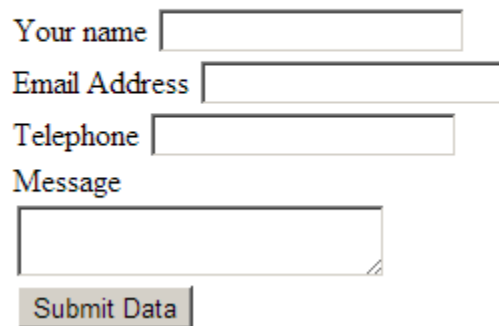
This is a simple example to make a form output nicer.

Consider the HTML document with a form:

```
<!DOCTYPE html>
<head><title>Using CSS to make form output nice</title></head>
<body>
<form>
  <div>
    <h1>Registration Form :</h1>
    <div>Your name <input id="name" type="text" name="name" ></div>
    <div>Email Address <input id="email" type="text" name="email"></div>
    <div>Telephone <input id="tel" type="text" name="tel"></div>
    <div id="msg">Message<br><textarea id="message" name="feedback">
    </textarea></div>
    <button type="Submit" id="btn" name="mybutton"> Submit Data </button>
  </div>
</form>
</body>
</html>
```

Output:

Registration Form :



The following CSS attempts to make it nicer looking:

```
<style>
form{
  background-image:url("swirl.gif");
  margin:auto;
  width:550px;
  height:450px;
  font-family: Tahoma, Geneva, sans-serif;
  font-size: 14px;
  font-style: italic;
  line-height: 24px;
  font-weight: bold;
```

```

    color: blue;
    border-radius: 10px;
    padding:10px;
    border: 1px solid #999;
    border: inset 1px solid #333;
    box-shadow: 2px 2px 8px blank;
}
input {
    width:375px;
    display:block;
    border: 1px solid #999;
    height: 25px;
    box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
}
#message {
    width:375px;
    height:150px;
}
#msg{
    display:block;
}
#btn {
    display:block;
    width:100px;
    margin-right:0px;
    background:#09C;
    color:#fff;
    font-family: Tahoma, Geneva, sans-serif;
    height:30px;
    border-radius: 15px;
    border: 1px solid #999;
    cursor:pointer;
}
</style>

```

Most of the CSS features here are familiar to you. There is one new property here, `cursor:pointer`; This makes the mouse pointer a hand when it comes on the button.

Study the stylesheet.

New output:

A registration form with a light blue wavy background. It contains four input fields: 'Your name', 'Email Address', 'Telephone', and 'Message'. The 'Message' field is a larger text area. A 'Submit Data' button is at the bottom left.

Registration Form :

Your name

Email Address

Telephone

Message

Submit Data

19. Customizing error messages

You saw default error messages when the user inputs incorrect data in form elements. We can customize the error messages using JavaScript. This involves capturing events and writing event handling functions. JavaScripts events are a big topic. We will skip it. Let us just use the HTML5 built-in constraints.
