Lab 9
Time Shift: Conditionals

Now that you know the basics of working with JavaScript, we'll work on a more complicated web page that will give us a chance to see how web forms and JavaScript programming can work together. In the process, we'll also get some practice with conditionals. We'll see how to use HTML to produce the graphical user interface for a web-based program, and we'll write JavaScript code to perform the computations the program requires.

Vocabulary

All key vocabulary used in this lab are listed below, with closely related words listed together:

tag, attribute
form element
variable, declaration, assignment
conditional
Boolean expression

Post-lab Questions

Write your answers after completing the lab, but read them carefully now and keep them in mind during the lab.

1. What might be an advantage of forcing the user to input times using drop-down lists instead of text boxes?

   What might be a disadvantage, either for the user or the programmer?

2. How are text boxes similar to variables. How are they different?

3. Describe two ways we've seen so far that JavaScript code in a web page can be executed.

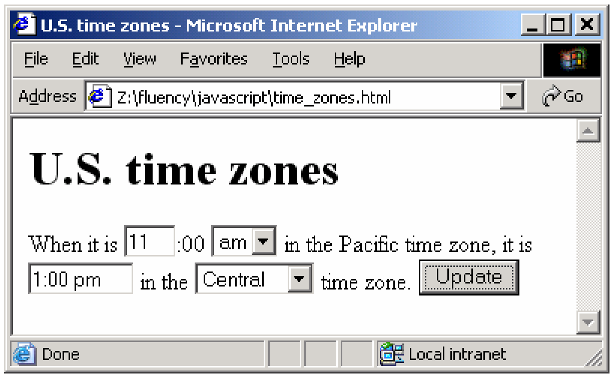
4. When you're finished with this lab, you'll end up with quite a bit of code stuck in `input` tag for the Update button. In another lab, we'll see how we can use functions to more cleanly separate the JavaScript from the HTML. Why do you think this separation might be desirable?

Discussion and Procedure

Part 1. JavaScript with web forms

In the last lab, we put a `<script>` section in an HTML file, and opening that web page in a browser caused the code to run immediately. In this lab, we'll see how we can write JavaScript code that doesn't run immediately when the page opened, but instead runs when the user clicks on a button, enters a value in a text box, or otherwise performs some action. Roughly speaking, we'll be using JavaScript to tell the page how to react when the user interacts with different parts of the page.

We'll start with a few, smaller exercises, but at the end of this lab, you'll create a page that can take a time from your local time zone and show you what time it would be in another U.S. time zone. The example below is shown for the Pacific time zone.


U.S. time zones - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address Z:\fluency\javascript\time_zones.html

U.S. time zones

When it is 11 :00 am in the Pacific time zone, it is 1:00 pm in the Central time zone. Update

Done Local intranet

Let's start with learning how to place text boxes, drop-down lists, buttons, and other kinds of form elements on a web page. Strictly speaking, this part of the lab is really more about HTML than about JavaScript. In general, form elements allow users to interact with web pages beyond just clicking on links. They are the *input* mechanism for interactive web pages designed using JavaScript or another scripting language.

HTML offers a wide variety of form elements, so we'll only cover a few of the more commonly used ones here: the text box, the button, and the drop-down list box.

1. *Start a new web page file* `time_zones.html`. Refer to lab 4 for the basics.

Open the file in a web browser so you can reload and see the results after each step.

2. *Add a* **`<form>`** *section.* In the **`<body>`** section, add a **`<form>`** section. In the opening tag, include the name attribute as shown below.

```
<form name=timeZones>

</form>
```

At this point, nothing will appear in your browser, because the form itself is empty. Form elements must be placed inside a **`<form>`** section, rather than just mixed in with the rest of your HTML. We are also giving this form a name, **`timeZones`**. The reason for this will become clearer later, but we will use the form name in JavaScript code to access its form elements, e.g., to get whatever text the user has entered in a text box.

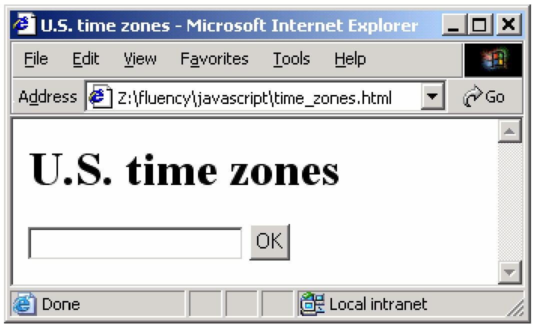
3. *Add an OK button to the form.* Adding a button only requires a single **`<input>`** tag with a few attributes, as shown below. The **`value`** attribute is what sets the text appearing on the button. At this point, the button isn't set up to do anything, so clicking it will have no effect.

```
<input type=button value="OK" />
```

4. *Add a text box to the form before the button.* Text boxes are similarly simple, but we'll use a **`name`** attribute, which we'll use to get what the user entered from inside our JavaScript program.

```
<input type="text" name="hoursBox" />
<input type="button" value="OK" />
```

Your web page should now look something like this:


U.S. time zones - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Z:\fluency\javascript\time_zones.html
Go
Done
Local intranet

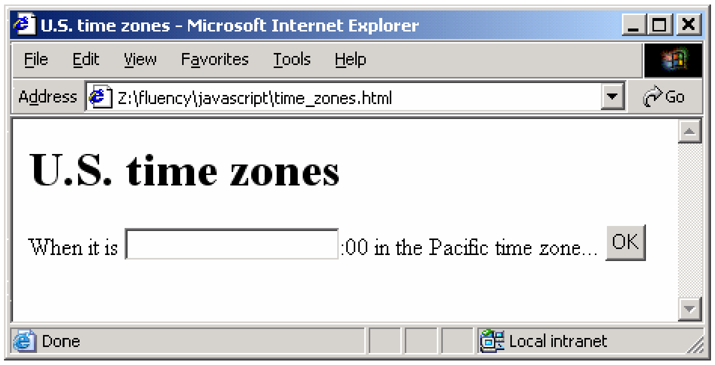
U.S. time zones

OK

5. *Add a* **`size`** *attribute with a value of* **`6`** *to the text box's* **`input`** *tag.* Did this change the way your web page appears when it is loaded? If so, how?

It's good practice to provide the web page user with some idea of what they're expected to do with the text boxes and other form elements.

6. *Add explanatory text around the text box.* Let's start building the user interface for our time zone page. Add some text on either side of the text box's **`input`** tag so your page ends up as shown below. Use the time zone you actually live in.


U.S. time zones - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Z:\fluency\javascript\time_zones.html
Go
Done
Local intranet

U.S. time zones

When it is :00 in the Pacific time zone... OK

Time zone reference. Not sure what time zone you live in? Want to know what time the U.S. government says it is in your town? Check out the web site time.gov, which is run by the National Institute of Standards and Technology and the U.S. Naval Observatory. You can also find some interesting historical facts about timekeeping on their site.

7. *Resize the* **`hoursBox`** *text box.* Since the user is only expected to enter at most two digits in the text box, we should make it a lot shorter. You can do this by adding a **`size`** attribute to the text box's **`input`** tag. Set the size to **`2`**.

Part 2. Putting some JavaScript behind the form

We've seen how to set up a very simple form with a text box and a button, but at this point, nothing happens when the user enters something in the box or clicks the button. Generally speaking, so far, we've only described what the page looks like using HTML, but we haven't encoded what the page should actually do. For this, we need a scripting language like JavaScript.

In the previous lab, we saw how to add JavaScript code to your HTML so that the code is executed as soon as the page is loaded. While this is generally useful, this isn't what we're looking for here. In this part of the lab, we'll see how we can set up the form elements with short sections of code that run when the user interacts with them in some way (e.g., clicking). For instance, we'll start by "attaching" some code to the button we have on our page. To get warmed up, we'll just set up the button so that it pops up an **`alert`** dialog box when it is clicked, and doing this is about as simple as it sounds.

8. *Add an* **`onclick`** *attribute to the button's* **`input`** *tag as shown below.* This should look a little strange to you, since we're sticking a piece of JavaScript inside an HTML tag, but this is the way to set up the button so that it runs the alert

code each time the button is clicked.

```
<input type=button value="OK"
onclick='alert("OK clicked!");' />
```

Note that we've put the line of JavaScript in single quotes. Why do you think it's necessary to use single quotes instead of double quotes in this case?

9. *Reload the page.* Does anything different happen when the page is reloaded (but before click the button)?

   What happens when you click the button?

In general, you could put any amount of valid JavaScript code in the value of the **`onclick`** attribute. The example code in Chapter 18 shows several form elements with more than a few lines of code embedded in their tags.

Now that we've seen how to build a web form and put JavaScript code behind it, we're ready to start on a "first draft" of the time zone web page in earnest. Before we begin, save your page.

Part 3. Version 0 of the time zone page: Input validation

Even though the time zone web page might not seem to complicated to you, it's important to use an intelligent strategy for producing it. The recommended strategy for any programming task is to build up your project in small steps. This process ensures that you will make steady progress (rather than getting stuck and frustrated) toward a working solution. As we step through the process toward completion for the time zone web page, you'll see how this strategy works in more detail.

To begin with, identify a reasonable, short-term goal: a simpler version of the final project that should be considerably easier to complete than the whole thing all at once. In the case of the time zone page, let's be very conservative and make our first goal a web page that doesn't actually do any time-zone computation. However, it will check what the user enters in the **`hoursBox`** text box to make sure the input is valid (i.e., make sure it makes sense). Input validation is frequently the first thing that programs do. After all, why bother proceeding with computation using the input if it doesn't make sense?

First, let's consider what we mean by valid input for this text box. Since we eventually plan to give the user the option of am or pm, we expect the user to enter an integer from 1 to 12. By using a conditional and some handy JavaScript tricks, we can check what the user enters in the text box and use **`alert`** to give the user instructions if they don't enter valid input. All of the code we write will be in the **`onclick`** value for the button. To get you started, we'll provide this code for you, but you'll be writing most of the remainder of the code for the time zone page.

10. *Change the button's* **`input`** *tag as shown below.* Note that we've changed the **`value`** attribute so the button says "Update" instead of "OK."

```
<input type=button value="Update"
onclick='
var hours =
  parseInt(document.timeZones.hoursBox.value);
if (hours < 1 || hours > 12) {
  alert("Please enter an integer from 1 to 12.");
  return;
}
// using alert to display input as reality check
alert("Input " + hours + " is valid.");
' />
```

11. *Reload the page and for each of the values listed below, try typing it in the text box and clicking the Update button.* Report what happens for each of the values.

    value: **`10`**

    value: **`1`**

    value: **`16`**

    value: **`-2.1`**

    value: **`11.53`**

    value: **`5hey`**

    value: **`hey`**

You've already seen conditionals in the Fluency textbook, but the **`parseInt`** line probably looks a little mysterious to you. First, you should recognize that this line declares a variable called **`hours`**. What's being stored in this variable? Two things are happening on the right side of the assignment here (i.e., on the right side of the **`=`**). First, we're retrieving the value inside the `hoursBox` text box. Look inside the parentheses after `parseInt`, and you'll see how we're using the form name (`timeZones`) and the text box name (`hoursBox`) to get the value:

```
document.timeZones.hoursBox.value
```

Reading from left to right, we're asking for the current HTML `document`'s `timeZones` form's `hoursBox` form element's `value`. Kind of like an Internet hostname, this identifier goes from general to specific, with periods separating the parts of the name. (Now do you see why it was important to give the form and the text box `name` attributes?)

What does the `parseInt` do? This is a handy JavaScript way of converting whatever the user enters in the text box into an integer value. For instance, if the user enters `11.53`, JavaScript's `parseInt` will convert that to just `11`, dropping the decimal part. If the user enters a number followed by some letters (e.g., `5hey`), `parseInt` will drop off the letters and give us just `5`. What `parseInt` doesn't do is make sure the number entered falls within a certain range, which is why we use the conditional to check this ourselves after the `parseInt` line.

The `return` that executes if the user's input is out of range simply causes the code to stop executing. In other words, this is what prevents the rest of the code from executing if the user's input was not valid. (Note that the second dialog box never appears if the input is out of range.)

When it comes to readability, humans first and computers second. It's important to remember that you and other people will be reading your code just as often as your computer will be "reading" it as it executes it. Writing your code so that it's easily readable and visibly well-organized will, in fact, help you quickly notice and fix bugs and simplify adding new code later. You should pick variable and HTML tag names sensibly. Instead of `k`, `x`, `y1`, `z3`, and other cryptic, short names, consider meaningful names like `hoursBox` and `ampm`. It's perfectly OK for an HTML tag or even a line of JavaScript code to be split across multiple lines in your file, which can make it considerably easier to read and edit. Indentation can be a helpful visual cue for split lines (like the `parseInt` line above) as well as for blocks of code in conditionals (also shown above). Finally, adding notes to yourself in the form of comments (text preceded by a double slash, `//`) can help you and others understand your code.

We can extend the input validation even further with another built-in JavaScript trick, `isNaN`. `NaN` stands for "Not a Number," and it works hand-in-hand with `parseInt`. If the user's input doesn't even remotely resemble an integer (e.g., a bunch of letters or words), the `parseInt` line will store the special value `NaN` in the `hours` variable. In that case, it doesn't even make sense to try to compare it to 1 and 12 to check if it's in the valid range.

12. *Add the condition* **`isNaN(hours)`** *to the Boolean expression, making it the first of the three conditions that are checked.* We'll leave it to you to decide whether you should use or (**`||`**) or and (**`&&`**) when adding this condition.

The user needs to provide two other pieces of input: whether the time is am or pm and the time zone for which the converted time is to be computed. These will be added in the form of drop-down lists, which are a little more complicated than text boxes and buttons to write in HTML. We'll provide the HTML for the am/pm list and leave the time zone list to you.

13. *Add the am/pm drop-down list by adding the HTML below to the* **`timeZones`** *form.* Note that drop-down lists are added using a **`select`** section, rather than a single **`input`** tag. Inside the **`select`** section, you use single **`option`** tags for the individual selections to be included in the drop-down list.

```
<select name=ampmList>
  <option value="am">am
  <option value="pm">pm
</select>
```

Now you can use **`document.timeZones.ampmList.value`** to retrieve the current selection in this drop-down list.

14. *Replace the last* **`alert`** *line with the code below to test that your drop-down list is set up properly.*

```
var ampm = document.timeZones.ampmList.value;
alert("Input " + hours + ":00 " + ampm
      + " is valid.");
```

Explain why validation of input for the am/pm drop-down list is not necessary, as was the case with the hours text box.

15. *Add a text box for the program's output, the computed time in the selected time zone.* Make it about size 8 and name it **`shiftedTimeBox`**.

This text box will be used to display the result of shifting the time to the selected time zone. So far, we've only seen how we can retrieve the value in a text box, but you can also assign a value to a text box. Try adding the assignment below to the very end of your button **`onclick`** code.

```
document.timeZones.shiftedTimeBox.value =
  "not sure yet";
```

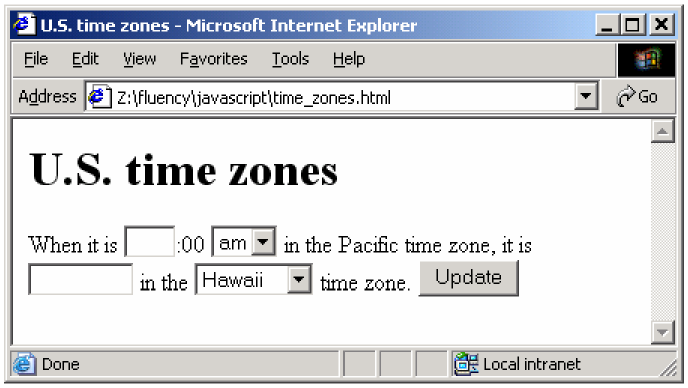
Eventually, we will add code to compute the shifted time and replace the right size of the assignment with the correct expression to output that.

What happens when you reload the page, enter a valid hour value, then click Update?

16. *Add a second drop-down list for time zone selection.* Use the am/pm list HTML above as a model. Name this new list **`zoneList`** and provide the choices Hawaii, Alaska, Pacific, Mountain, Central, and Eastern, which are the major time zones covering the fifty states.

17. *Add code to retrieve the selected value from* **`zoneList`** *and store it in a variable named* **`zone`**. Use the analogous code for the am/pm list above as a model.

Your web page should now look like this:


U.S. time zones - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Z:\fluency\javascript\time_zones.html
Go
Done
Local intranet

U.S. time zones

When it is :00 am in the Pacific time zone, it is in the Hawaii time zone. Update

Although it doesn't compute the shifted time, it does do input validation, which, as you saw, is no small task.

Part 4. Version 1 of the time zone page: Ignoring am/pm

Our next goal is a web page that does part of the time conversion: it will give the time in the selected time zone, but it won't say whether the shifted time is am or pm, which, as you'll see, is a little tricky to figure out.

Before we try to write code to solve the problem of shifting time to a different time zone, make sure you can do it yourself! After all, how can we expect to write precise instructions for a computer to do this if we're the slightest bit unclear on how to do it ourselves? Start by figuring out what the time difference is between your local time zone and the other time zones. For instance, if you're in the Pacific time zone, the difference to the Eastern time zone is +3 hours, and the difference to Hawaii is –2 hours. Fill in the table below for your local time zone. The time zones are listed in order from east to west, and each one differs from the next by one hour.

time zone	**difference from local time**
Hawaii	
Alaska	
Pacific	
Mountain	
Central	
Eastern	

This table simplifies computing shifted time. All you do is take the local time and add the difference. If the result is less than 1 or greater than 12, you need to do a little arithmetic to end up with a valid time, but that's all.

For example, 1:00 Pacific time is 11:00 Hawaii time, because the difference is –2 hours. To get this, we add –2 to 1, which yields –1. Since –1 is not a valid time, we correct it by adding 12, and we get 11.

So the algorithm for shifting time (ignoring am/pm for now), is something like this:

1. Depending on the selected time zone, determine the difference from local time.
2. Add this difference to the hour part of the local time.
3. If the result is less than 1, add 12; on the other hand, if the result is greater than 12, subtract 12.

Which of these steps do you think will require a conditional?

Based on this planning, try putting the above algorithm in code. We suggest you use at least a couple additional variables: one to store the time difference, one to store the shifted time (in hours). At the end, display the shifted time in the text box **`shiftedTime`**. You might want to leave out Step 3 above and get the time-shifting computation without the correction working first. (Remember, small steps!)

Don't forget to test your page with a variety of times to verify that it's working properly. Especially once you get the correction (Step 3) working, test that it works for times that you know will require correction (as in the Pacific-to-Hawaii example above).

Part 5.   Version 2 of the time zone page

Recognizing when you need to add or subtract 12 to correct a shifted time is easier than recognizing whether shifting the time changed it from am to pm or vice versa. In general, the am/pm must be "flipped" (am to pm or pm to am), when the time shifts forward past 11:00 or shifts backward past 12:00. Expressing this as a Boolean expression requires some careful thought.

So you can concentrate on this complicated Boolean expression, start by writing the code to "flip" the value of the **`ampm`** variable. Test that it works properly by including the value of **`ampm`** in the output in **`shiftedTimeBox`**.

The additional code you need to add for this final part of the lab will look like this, leaving some blanks for you to fill in:

```
if ( Boolean expression ) {
   code to flip ampm
}
```

That's not a lot of code, but it will require some thinking.

See next page for grading rubric and extra-credit options.

Extra Credit Activities

- Make a copy of your page by saving it as **`fit100/lab09/time_zones_function.html`** Move your function from the **`onclick`** event in the input element to a separate function that is called from the **`onclick`** event. Call the function **`convertTime()`** and declare it in the **`<head>`** section of the html page.

- Make a copy of your page by saving it as **`fit100/lab09/time_zones_more.html`** Add two more time zones to your GUI and provide the additional coding.

- Make a copy of your page by saving it as **`fit100/lab09/time_zones_military.html`** Modify your code to use inputs and outputs time in 24-hour format (also known as "military time") rather than using am/pm. Be sure to save it again when you're done and upload it to your Dante account.

- Make a copy of **`fit100/lab09/time_zones_military.html`** by saving it as **`fit100/lab09/time_zones_both.html.`** On the GUI give the user a choice of military time or am/pm. Be sure to save again when you're done and upload it to your Dante account.

Rubric

Item	**Points**
GUI with form section and all inputs (looks like screenshot in part 4)	5
Input validation: "Please enter an integer between 1 and 12" Test: 15, 0, text, blank	5
Test: 9 am to Eastern Time (ET), 9pm to ET • Time shift works going east • Am/pm flip works going east for both	5
Test: 2pm to Alaska Time (AT), 2am to AT • Time shift works going west • Am/pm flip works going west for both	5
Extra Credit: **`onclick`** calls a convertTime function in the **`<head>`**	5
Extra Credit: two more time zones added	5
Extra Credit: military times (24-hr clock)	10

Extra Credit: offers choice of military or am/pm time (works with both)	5

Turn-in at end of Lab

Submit your Lab 09 work through Catalyst Collect-It. For Lab 09 please turn in only a Word document or lab09.txt file with the URL for your html file, your name, UW NetID, Student ID No., and Lab section, and the answers to the Post-Lab Questions listed at the beginning of this lab.

Lab 9 Online Due Date: Monday, March 3, before 5:00 pm.

Upload only the Word or lab09.txt file to the Catalyst Collect-It Turn-in Area on the Course Calendar for this lab number. Do not continue working on the lab after the deadline or it will be considered late.

If you want to make further changes to the lab, copy the lab and save it as a different file name before uploading it. If you make changes to the file listed in the URL you submitted in your Word or .txt document, your lab will be considered late.