Worksheet 7: Web Development

Updated: 21st October, 2018

In this worksheet, we'll go all the way back to the start, and implement a simple calculator, but this time as a web page.

We won't worry about the server-side activity at this point, and instead simply store all the files locally. (For this kind of simple app, the server need not really do anything anyway except deliver the required files.)

You will need: A text editor and a web browser! We no longer need Android Studio, and there's no compiler either (at least not for what we're doing).

1. HTML

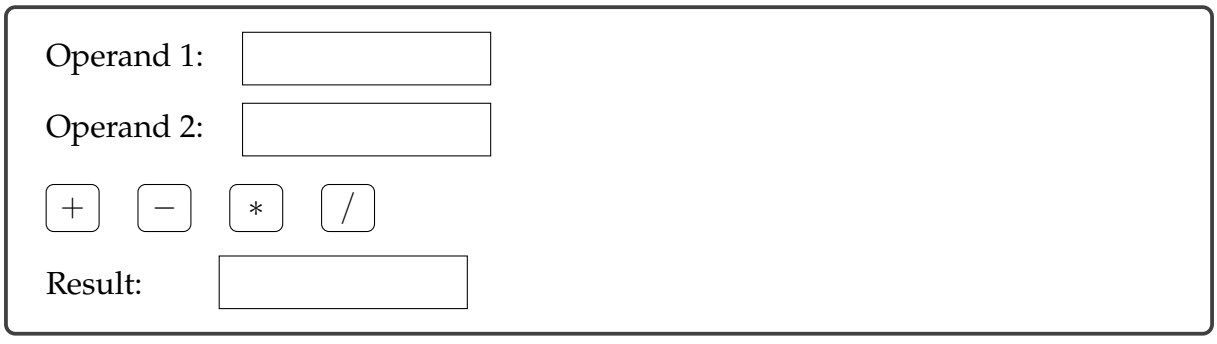
You should be able to construct an appropriate .html file from examples given in the lecture. In particular, you will need several `<input>` elements, each with a unique ID:

- Two for the operands. These should be of type "number".
- Four for buttons labelled "+", "−", "∗" and "/" (type "button").
- One for the result (type "text"). If you like, you can add a `readonly` attribute:

```
<input ... readonly />
```

(Note: we could actually just use a `<div>` or `<span>` for this purpose, since this doesn't need to be user-editable, but we'll go with `<input>`.)

Aim for a layout that resembles the following:



Check your work by loading it in a web browser.

(Ctrl-O should let you load a local file. You can also enter a URL starting with "`file://`", followed by the path to the .html file; e.g. `file:///home/username/work/myapp.html`.)

Warning: If your HTML is invalid, web browsers generally try to *guess* what you mean, rather than giving an error! This sounds nice, but really isn't.

In "developer tools" (F12), check whether the parsed HTML structure is actually what you wrote or not.

2. CSS

We're going to play with the mechanics of CSS. (Actually making a web page look nice can be quite difficult, and we won't be setting a terribly high standard for artistic talent here!)

First, create an empty .css file, and link the HTML to it (using a `<link ... />` tag as in the lecture notes.

Add rules to the .css file to accomplish the following:

(a) Make the whole `<form>` element[1], light blue with a blue border, and a sans-serif font. You can use the following CSS declarations as part of the rule:

```
border: 2px solid blue;
padding: 5px 10px;
background: #c0c0ff;
font-family: "Georgia", serif;
```

> **Note:** You can see the necessity of the "`padding`" property if you remove it. Padding refers to extra space *within* the border. If you give it two values (as above), the first is the vertical (top and bottom) padding, and the second is horizontal (left and right).

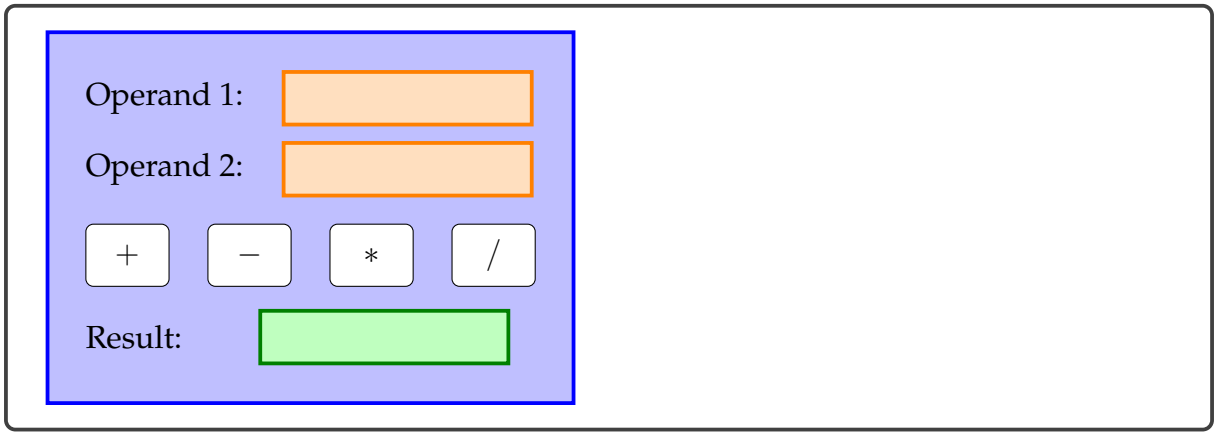
(b) Give all `<input>` elements a vertical and horizontal padding of `5px` and `10px`.

(c) Make the operand fields (only) light orange, with an orange border, using these declarations:

```
border: 2px solid orange;
background: #ffe0c0
```

(d) Make the result field (only) light green, with a green border. This will be the same as the above, except with "`green`" and "`#c0ffc0`".

The page should now look *roughly* as follows:



[1] You remembered to have one of those, right?

3. Hypothetical Selectors and Classes

The CSS selectors you had to use in the previous question were (hopefully) fairly simple. Just for a moment, let's consider some more complicated possibilities.

How could you make use of CSS selectors *and* the `class="..."` attribute in the following cases?

(a) You introduce a whole set of new operations (x^y, `log`, `sin`, etc.), and would like to assign different colours to different groups of buttons.

(b) You integrate the calculator into a much larger and more complex web page, which includes other `<form>` elements, and a lot of other `<input>` elements of various types. You *only* want the calculator style to apply to the calculator itself.

Note: try to come up with a simple solution that minimises the amount of "extra stuff" you have to write.

4. JavaScript and jQuery

Time to get back to the actual calculator, and make it work! (We could do this all without jQuery, but some jQuery practice will be useful.)

As you're completing this, you will *definitely* need to use the "console" panel in "developer tools" (F12). At the very least, this will show you all your JavaScript error messages. It will also show you any messages you output with `console.log("...")`.

You will need to do the following:

(a) Use a `<script>` element to import the jQuery library, as shown in the lecture notes.

(b) Create an empty .js file, and link the HTML to it using another `<script>` element.

(c) Within your .js file, use jQuery to make an event handler for the window "load" event. *The rest of your code must go inside this event handler!*

(d) Get references to the operand and result elements. We'll actually do this part without jQuery, and you may notice it looks a bit like the equivalent in Android.

Note: Without jQuery (and as shown in the lecture notes), this will look as follows:

```
let element = document.getElementById("...");
```

"`let`" just indicates a variable declaration. You may sometimes see the obsolete "`var`" keyword used the same way, but `let` is now the preferred approach, as `var`'s scoping rules are strange and often unhelpful.

We could alternatively use "`const`", which works like "`let`", except it doesn't allow re-assignment.

> **Note:** If you were to do this *with* jQuery, be aware that jQuery returns *sets* of elements, which are not the same as individual element objects, *even if* the selector matches only one element.
>
> However, you can treat the element set as an array, and thus use the index notation to get the actual element:
>
> ```
> let element = $("...")[0];
> ```

(e) Set a jQuery event handler for the "click" event for the plus button. The relevant data will be inside the "`value`" field of each `<input>` element. You simply need to add the operand values and set the result value accordingly.

> **Note:** `value` is a string, even if `type="number"`. (`type="number"` is only a suggestion to the browser that it should prevent non-numerical input, but it doesn't try very hard!)
>
> To convert strings to numbers in JavaScript:
>
> ```
> let num = parseFloat(str);
> ```

(f) If the values of the operands were not valid numbers, you'll see that the result is "`NaN`" (not-a-number). Rather than outputting this, we'd instead like to generate an error message.

To do this, you will need to invoke `isNaN()` to check the operand values, and if either is `NaN` then call `alert()` with an error message.

> **Note:** `NaN` has the unique and irritating property that *it is not equal to itself*; i.e. `NaN != NaN` and even `NaN !== NaN`.
>
> Unlike a lot of other weird results, this one is *not* actually JavaScript's fault, as this behaviour is described by the IEEE specification for floating-point numbers. The same thing happens in C, for instance, when working with floats and doubles.

(g) Finally, implement the other three buttons.

End of Worksheet