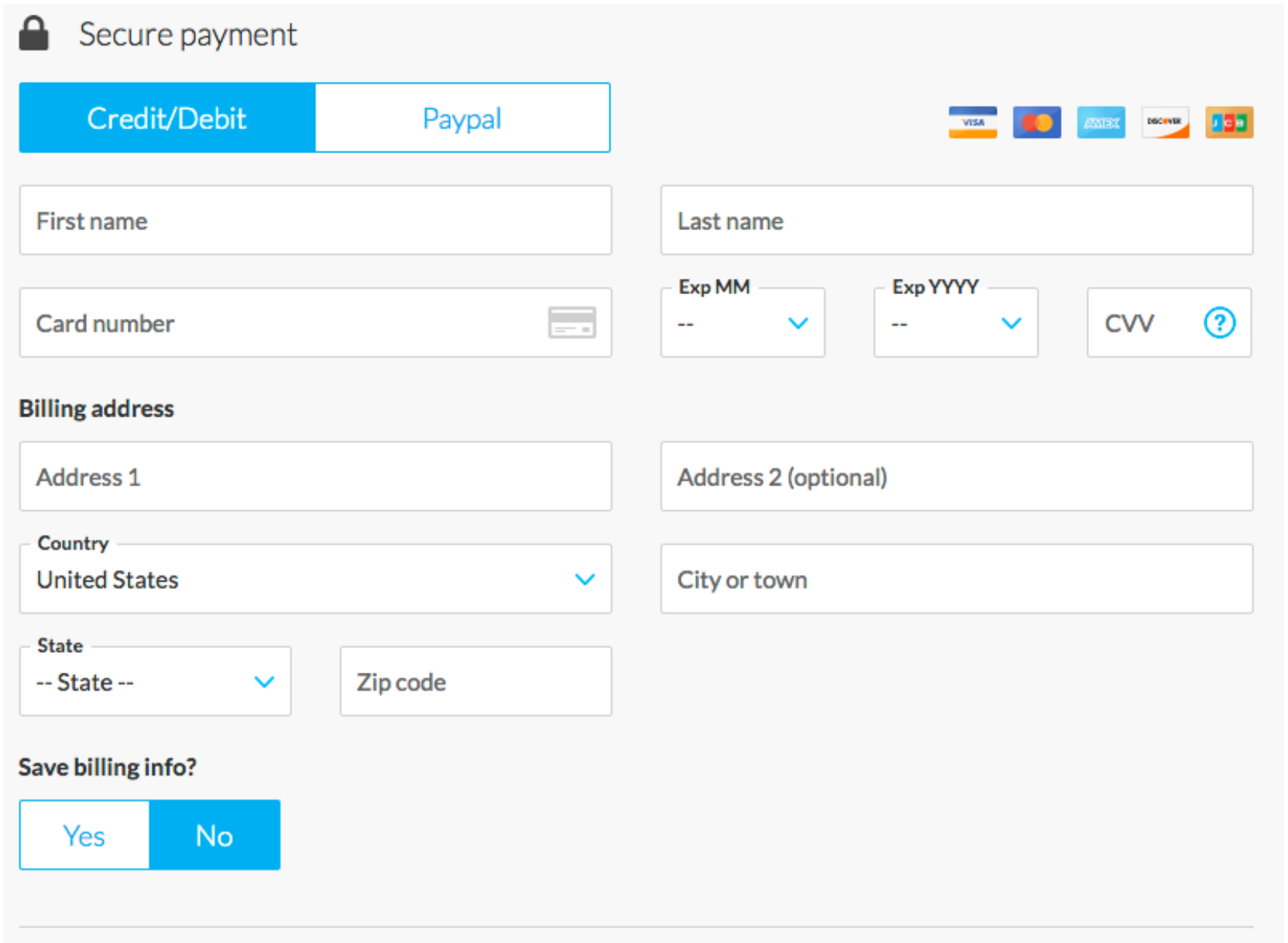

FORMS

FORMS


Secure payment
Credit/Debit
Paypal
First name
Last name
Card number
Exp MM
--
Exp YYYY
--
CVV
Billing address
Address 1
Address 2 (optional)
Country
United States
City or town
State
-- State --
Zip code
Save billing info?
Yes
No

WHAT FORMS DO

-Forms capture user input from the web and send it off to the backend of a website to be processed in some way

-Any time you put any content into a webpage, you are using a form

SIMPLEST FORM

```
<form>
  <input type="text">
  <input type="submit">
</form>
```

Forms have a lot of markup and are tedious to build - normalize.css spends most of its energy taming form inputs

<FORM>

```
<form>
  <input type="text">
  <button type="submit">
</form>
```

All forms have the form tag wrapping around them. Always include.

<FIELDSET>

```
<form>
  <fieldset>
    <input type="text">
    <button type="submit">
  </fieldset>
</form>
```

Fieldset wraps around a set of field inputs on the form. Optional but I like it.

<LEGEND>

```
<form>
  <fieldset>
    <legend>Form Title</legend>
    <input type="text">
    <button type="submit">
  </fieldset>
</form>
```

Legend is where you put the title of the form (optional). Rarely used but it has a soft spot in my heart.

<TEXTAREA>

```
<textarea placeholder="Write something
here"></textarea>
```

Creates a really big text area. Don't size it with HTML attributes, use CSS please! Don't be that guy.

<SELECT>

```
<select id="select">
  <option value>Choose</option>
  <optgroup label="Option Group 1">
    <option value>Option 1</option>
    <option value>Option 2</option>
  </optgroup>
</select>
```

Selects are for those lists you click on to select an item.

<LABEL>

```
<form>
  <fieldset>
    <legend>Form Title</legend>
    <label for="field1">New Field</label>
    <input type="text" id="field1">
    <button type="submit">
  </fieldset>
</form>
```

Label is the tag for telling a user what label of a field is. Note the 'for' attribute - what is that doing?

<INPUT>

```
<form>
  <fieldset>
    <legend>Form Title</legend>
    <input type="text" id="field1">
 </fieldset>
</form>
```

Input is the workhorse of the form world - it can do a great many things.

<INPUT>

Huge list of input types (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input)

Focus in on a couple important types:

```
<input type="checkbox" id="field1">
<input type="radio" id="field1">
```

CHECKBOXES

```
<div class="checkbox">
  <input type="checkbox" id="checkbox2">
  <label for="checkbox2">5</label>
</div>
```

You have to put the labels after the checkbox for them to make any sense. I usually group them like this.

RADIO BUTTONS

```
<div class="radio">
  <input type="radio" id="radio1" name="radio-group">
  <label for="radio1"><span>Good</span></label>
</div>
<div class="radio">
  <input type="radio" id="radio2" name="radio-group">
  <label for="radio2"><span>Bad</span></label>
</div>
```

Radio buttons are very similar to checkboxes but function as a group with the 'name' attribute.

STYLING FORMS

NEW SELECTOR

```
input[type=text] {
  margin: 25px 0;
}

input[type="text"] {
  margin: 25px 0;
}
```

If you want to target a particular attribute in HTML, use the brackets. Both syntaxes are the same, no spaces.

SAME AS IT WAS

```
#style-me {
  margin: 25px 0;
}

input[type="text"]#style-me {
  margin: 25px 0;
}
```

Generally, I style forms by targeting IDs, just like JS fun times. Why not classes? Form elements usually need unique IDs for labels to work.

:FOCUS

```
textarea:focus {
  outline: none;
}
```

There may be cases where you want to style what it looks like when a user is active on a field - use the :focus pseudo-selector. Outline property is helpful here.

STYLING NOTE

In general - don't style the text boxes, checkboxes and radio buttons themselves. You can do this but the techniques will make you want to jump out of window.

If you want to try this, let's talk offline - it's quite a bit of work and takes a lot of cross browser testing.

CODE ALONG

Assignment #2

ON THE SUMMIT OF BASIC JS


FROM HERE
You've got the basics of JS now, up to you to carry it further. We're here to help if you want to go there.

FORM SUBMISSIONS

FORM ACTION

```
<form action="send.php" action="POST">
</form>
```

The action attribute tells the browser where to send the form information to. This is usually a backend programming language - like PHP, Ruby or Python.

EASY OPTIONS

Most simple sites do email contact forms. Some options for submitting that data:

Cool easy option - emails you!: https://formspree.io

Sign-up forms that capture the data: http://mailchimp.com/

FORM SPREE

This is all you need on your site to get emails (this form will post email submissions to your address):

```
<form action="https://formspree.io/
your@email.com" method="POST">
  <input type="text" name="name">
  <input type="email" name="_replyto">
  <input type="submit" value="Send">
</form>
```

HARDER OPTIONS

1- You have to use a backend technology (Ruby/Rails, Python, PHP, C#, .net, node.js, etc)

2- You can send transactions as GET or POST methods.

```
<form action="file.php" method="post">
</form>
```

GET VS POST

1- GET transactions take the form of query parameters and get appended to URLs (insecure).

2- POST transactions get interpreted by their destination and sent back to the host (secure).

GET METHOD

The GET method appends transaction data to the URL string you provide (this can be a local file as well):

```
<form action="http://www.example.com" method="get">
  <div>
    <label for="state">What state do you live in?</label>
    <input name="state" type="text">
  </div>
  <div>
    <label for="city">What city do you live in?</label>
    <input name="city" type="text">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

GET RESULTS

The GET method will append results to your URL (remember the action can also be a local destination):

```
<form action="http://www.example.com" method="get">
  <div>
    <label for="state">What state do you live in?</label>
    <input name="state" type="text">
  </div>
  <div>
    <label for="city">What city do you live in?</label>
    <input name="city" type="text">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

=> http://www.example.com/?state=yourstate&city=yourcity

POST METHOD

The POST method sends data for processing and then feedback (more common):

```
<form action="http://www.example.com" method="post">
  <div>
    <label for="state">What state do you live in?</label>
    <input name="state" type="text">
  </div>
  <div>
    <label for="city">What city do you live in?</label>
    <input name="city" type="text">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

POST RESULTS

The POST method will complete a HTTP transaction to a location. It is up to the remote location to process and send back appropriate results.

HTTP transaction results come in the form of HTTP codes.

HTTP CODES GUIDE

HTTP codes are three digit numeric values that indicate the state of a internet transmission.

```
Common HTTP Codes:
=> 200 (success)
=> 301 (moved)
=> 400 (bad request)
=> 403 (forbidden)
=> 404 (not found)
```

Full list:
http://www.restapitutorial.com/httpstatuscodes.html

HTTP CODES DEBUG

Your browser doesn’t show HTTP codes or the transactions themselves, only the results. To see the transaction in action, use Chrome Dev Tools > Network tab.

CODE ALONG

Assignment #4

FORM DATA RETRIEVAL

AFTER SUBMITTING...

Your data gets attached to two special global variables that can ONLY be read by backend scripts:

```
// If you used post
$_POST['city'];
$_POST[‘state'];

// If you used get
$_GET[‘city’];
$_GET[‘state’];
```

KEY IDEA

Form submissions are the gateway of data transfer between front and back end systems.

FOR LATER

Run Assignment #5 on a server


APIS

WHAT IS AN API?

API stands for Application Program Interface.

It’s the mechanism that dispenses structured data to webpages but it’s hard to make sense of.

Think of the Internet as a series of buildings…some big, others small.

Each building has a door


And doors
require
keys…

KEY IDEA

API’s are doorways into different website’s data. For most sites, you’ll need a key to get inside (often just called API keys).

HOW TO ACCESS AN API

1) Simplest: you'll have to get an API key to perform a transaction with an API.
2) Moderate: If the API gets/puts data into an account, you will likely need account information as well.
3) Tough: You may have to enter a username/password using OAuth protocol.

Once you unlock the door, you don't know what will be inside (or how many rooms there are).

WHICH ROOM?

Continuing this metaphor, API “rooms” are called endpoints. Each API has different endpoints that return different data once you transact the key/user/OAuth sequence.

READ THE DOCS!

Every API is unique - you will have to explore its documentation to figure out which endpoint you want to use.

LETS EXPLORE DOCS

This is a huge public catalogue of APIs (but not exhaustive, there are thousands more):

https://github.com/toddmotto/public-apis

THE YELP API

We're going to use the Yelp v3 'Fusion' API to experiment with getting data from a remote source:

https://www.yelp.com/developers

INSTALL THE YELP API

https://github.com/Yelp/yelp-fusion/tree/master/fusion/node

MAKING AN API CALL

We’re going to keep using jQuery and one of it’s most useful parts: .ajax();

.AJAX() - BASIC

```
// Attach to .ajax() to the global
jQuery instance
$.ajax({
  type: ‘POST',
  dataType: 'json',
  url: ‘linktoprocessingscript.php’,
  data: variableGoesHere,
  success: function (results) {
    // do something
  }
  error: function (results) {
   // do something else
  }
});
```

JS OBJECTS

API endpoints will almost always return JSON in the form of an object.

```
{
  “data”: “car”,
  “make”: “ford”,
  “model”: “focus”
  “details”: {
    “color” : “blue”,
    “mileage” : “54019”
  }
}
```

OBJECT LITERAL

You'll want to turn the response from the API into a variable. This gets called an object literal.

```
var myDataObject = {
  "data": "car",
  "make": "ford",
  "model": "focus"
  "details": {
    "color" : "blue",
    "mileage" : "54019"
  }
};
```

OBJECT REFERENCE

You can then access parts of the object with a convenient syntax:

```
var myDataObject = {
  “data”: “car”,
  “make”: “ford”,
  “model”: “focus”
  “details”: {
    “color” : “blue”,
    “mileage” : “54019”
  }
};

myDataObject.make; // ford
myDataOject.details.color // blue
```