Task B: Create an HTML form with a text field, a password field, an email field and other optional fields. Use the `POST` method to transfer the form data. The link for the form action will be released later and will be intimated to you through Piazza. On submitting the form, the server will check the correctness of the inputs and echo back all the values it received. You are not allowed to use JQuery or any external libraries for this. The code needs to be in native JavaScript. You will need to conform to the following specifications:

- The number of characters in the text field should be in the range 6-50, both inclusive. Use `"text"` as the `name` of this field
- The password should not be visible and thus suitably masked. It should be 8-20 characters in length with at least one special character from the set `{$, !, @, _}`. Use `"pass"` as the `name` of the element
- Use `"email"` as the `name` of the email field. The email address should conform to the following specifications:
  - It should be of the form **local-part@domain-part** where
  - *local-part* can contain only alphanumeric characters in addition to the special characters `{., _}` with length of the string >2
  - *domain-part* should contain only alphabetical characters other than a period(`"."`). There should be at least one period(`"."`). Each substring delimited by the `"."` characters in this part should have at least 2 alpha-characters between them. The substring of domain-part before the first `"."` should have at least 3 alpha-characters

Having done this, you are expected to secure the form against the most basic type of XSS attack.

A simple XSS attack is when JS code is injected between the `<script>` and `</script>` tags. For example, if the browser receives `<script>alert("Hacked")</script>` as in input, then an alert box will pop up showing the message `Hacked`.

The problem arises because the code between the `script` tags is identified by the browser as JS and it executes that. A solution to this is to make the `script` tags lose their meaning, i.e., the browser does not treat them as structures (or 'structs') carrying code within them but instead just simple pieces of text. A simple way to do that is to encode special HTML characters. Figure out what they are and how to encode them in a HTML safe manner. Do not use any predefined functions for this. You have to write code for this.

Extra credits: Try implementing on the fly form validation as the user is filling out the form. Refer to the form on this page for example. You can earn up to 10 points for attempting this task. Try filling in fields of the form incorrectly to see the error messages.

On the go: Accordion

Since Pokémon Go is literally on the streets these days, let's walk in that direction.

Task C: Create a self-stylized accordion with the following specifications:

- Create at least 4 collapsible tiles
- Header of each section should contain the name of a Pokémon
- On expanding the section, the type, final evolution state of the Pokémon (if any) and four most common attacks of the Pokémon should be made visible

Just to reiterate, you are not allowed to use any existing themes or templates for this. Read the references provided and figure out how to go about doing this. Here are some screen shots for your reference. (Note: If you have no idea about Pokemons you can use your the 4 largest metros. Interpret 'attack' and 'evolution state' in your own way but be prepared to defend your interpretation.)