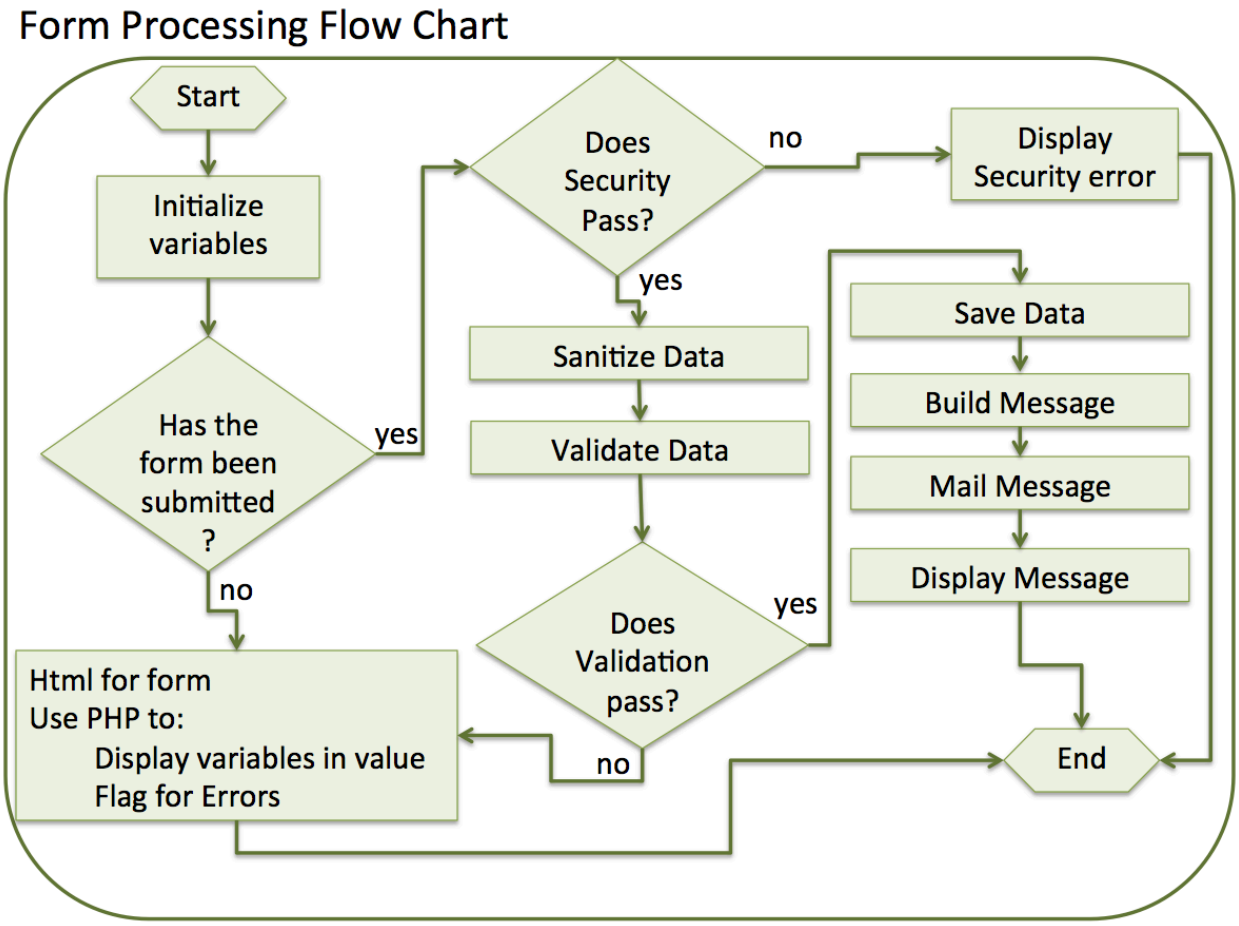
Chapter 7 Form Processing – Validate Data



Once our data has been sanitized we can start working with it. Our next step in the flow diagram is Validate Data which means checking the value of your data to make sure the value is acceptable. This corresponds to Github commit seven. We cannot check for everything but we can check for many things:

1. Is there something in the box?
   ```
   if ($password == "") {
   ```

2. Is the value text or numbers or both?
   ```
   if (!verifyNumeric($total)) {
   if (!verifyAlphaNum($password)) {
   ```

   Where do these functions come from? Hang on I will show you soon.

3. Does the value fall within a range (ie. 0 to 100? In this example the max length should be set to 3 even though 999 would still pass that simple test it would keep 1000 from getting in).

```
if ($grade < 0 or $grade > 100) {
```

4. Does the value contain certain values (ie. email address text@text.com, file extension [.jpg]) (where $ext has the file extension in it).
```
$extensions = array("jpg", "png");
if (in_array($ext, $extensions)) {
```

5. Does the value need a particular length (ie. password must be 6 characters long)
```
if (strlen($password) = 6) {
```

6. Does the value need a minimum length (does anyone have a name less than 2 characters?)
```
if (strlen($password) < 2) {
```

To check these things you need to make a function to do so. Here I have made several functions that you can use. Notice they all return true or false. Since we are interested in if they don't pass I have used the ! symbol in some of my if statements that call these functions. I put these functions in a separate file.

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// series of functions to help you validate your data. Notice that each
// function returns true or false
function verifyAlphaNum($testString) {
     // Check for letters, numbers and dash, period, space and single quote
     // only. added & ; and # as a single quote sanitized with html entities
     // will have this in it bob's will be come bob&#039;s
     return (preg_match ("/^([[:alnum:]]|-|\.| |\'|&|;|#)+$/", $testString));
}

function verifyEmail($testString) {
// Check for a valid email address
// see: http://www.php.net/manual/en/filter.examples.validation.php
     return filter_var($testString, FILTER_VALIDATE_EMAIL);
}

function verifyNumeric($testString) {
     // Check for numbers and period.
      return (is_numeric($testString));
}

function verifyPhone($testString) {
// Check for usa phone number
//  see: http://www.php.net/manual/en/function.preg-match.php
// NOTE: An area code cannot begin with the number 1, often when you type
```

```
// a number for testing you type 123 ... and it will not pass validation :)
     $regex = '/^(?:1(?:[. -])?)?(?:\((?=\d{3}\)))?([2-
9]\d{2})(?:(?<=\(\d{3})\))? ?(?:(?<=\d{3})[.-])?([2-9]\d{2})[. -]?(\d{4})(?:
(?i:ext)\.? ?(\d{1,5}))?$/';
     return (preg_match($regex, $testString));
}
```

You will notice that most of these functions use what is called a regular expression [http://www.regular-expressions.info/] which is a very powerful expression. You do not need to be able to write regular expressions but let's just dissect a simple one:

```
/^[a-zA-Z\s]+$/
```

1. / begins and ends the regular expression
2. ^ start at the beginning character
3. [matches a single character out of all the possibilities inside the brackets
4. a-z means any lower case letter from a to z
5. A-Z means any upper case letter A to Z (see character set table below)
6. \s means whitespace
7.] so only match letters and whitespace
8. + one or more of the previous are allowed (ie. you can have more than one letter)
9. $ Matches at the end of the string the regex pattern is applied to
10. So we match one more letters and whitespace for the whole length of the string, it checks each character one at a time. If one character is not a letter or a white space it returns false.

The basic structure of validating our data is more or less the same.

1. Perform check (ie if statement)
2. Keep track of an error message
3. Flag that there is a mistake
4. Repeat if needed (ie elseif)

Here is what we do for validating an email address (see how we repeat steps 1, 2, 3):

```
1. if ($email == "") {
2.     $errorMsg[] = "Please enter your email address";
3.     $emailERROR = true;
1. } elseif (!verifyEmail($email)) {
2.     $errorMsg[] = "Your email address looks incorrect.";
3.     $emailERROR = true;
   }
```

If a field is required we just ask if it is empty. Since we do not know how many mistakes a user may make we accumulate the error messages into an array. Then we can use a for

each loop to display all of them in an ordered list. This will display before the form and is for accessibility. A blind person will hear the form has the following mistakes then speak them in the order they are on the form. For this reason be sure to check for errors in the order the form objects appear on your form.

If the variable is not empty we perform the next check on it, which calls the function verifyEmail. Since verifyEmail returns true when it is good we use the ! operator as we are only interested in when it fails (remember this is the same as:

```
if (verifyEmail($email)== false)
```

If there are mistakes we want to redisplay the form. By putting the values in the form variables (when we sanitize the data) are form is setup to be sticky so that all the values the user typed in will display back on the form again. It is pretty neat logic and may take awhile for you to wrap your head around it.

Summary

We always need to check to see if the data we are getting is going be correct. Several things we can check

- if empty or missing
- Numbers, characeters
- check if value is within range [strlen($password) == 6]

Self Test Questions

1. **Empty Checker**
   Build a simple form with one element and a submit button. On submit, validate to make sure the data entered is not empty.

2. **Content Checker**
   Build a simple form with one element and a submit button. Create an array of strings to test against. On submit, validate to see if the submitted data contains a string from the array. Print “true” if so, “false” if not.

3. **Length Checker**
   Build a form with two elements and a submit button. Validate the two entries to ensure they are the same length. If they are not, print false;

4. **Email Validation**
   Create your own function that validates an email by simply checking if it contains an @ symbol. Use the strpos function as shown below:

```
$email = “myemail@email.com”;
if (strpos($email, '@') == true) {
   echo 'true';
}
```

Build a form with one element and a submit button, and validate the email.

5. **Password Validation**

Create a rudimentary password validation service using a form with one field and a submit button. The password should be stored as a variable, and the code should check to see if the entered password matches the stored variable and return true if so.