Validating User Input

> We have to make sure it's easy for users to know what kind of information we want from them and to enter it properly

- Minimize errors
- Correct type

We'll Explore the Following Topics

1. Types of validation
2. How to prevent validation errors
3. How to handle validation errors

Types of Validation

- Required Information
  - Some fields cannot be empty
- Correct Data Type
- Correct Format
  - Postal Codes, Credit, etc
  - Apply some automatic formatting if possible
- Comparison
  - Some fields depend on other inputs
- Range Check
  - Birth dates
- Custom
  - Username is not already taken

How to Prevent Validation Errors

How to reduce validation errors: provide the user with information about how to correctly fill out the form

- Sensible Default values

```
<input type="text" placeholder="Enter the height">
```

- Tooltips or pop-overs
- JS based mask
(___)—___—____
- List of options

How to handle validation errors

Notifying the user problems

- What is the problem?
  - Clear
  - Concise
- Where's the problem
  - Some kind of indication should be near the field
- How to fix
  - What the expected format/type

Where to perform validation

Client Side then Server Side

Client Side

- Reduce Server load, improver user experience
- Some validation is built in HTML5
- Additional can be done with JS

Validation on server side is the most important
Because client side can be maliciously used (bypassed)

Validation in PHP

`$_Get` or `$_POST` is used to get the variables

To validate:

- Email, numbers, URLs: `filter_var`
- Dates: `date_create_from_format` , `date_get_last_errors` , or `checkdate`
- Any other string: `preg_match` to check using regular expression

`filter_var` function can check certain inputs' valid format

- emails
- numbers
- URLs
- etc

```
```

```
filter_var(mixed $value, int $filter = FILTER_DEFAULT, array|int $options = 0): mixed
```

- value — a value to filter
- filter — the filter to apply
  - FILTER*VALIDATE constants*
  - *sanitization filter*
  - *FILTERSANITIZE*
  - FILTER_UNSAFE_RAW
  - FILTER_CALLBACK — a custom filter

  Success returns filtered data
  false is returned on failure
  FILTER_NULL_ON_FAILURE returns null

examples

```
var_dump(filter_var('bob@example.com', FILTER_VALIDATE_EMAIL));
```

```
if(filter_var($_POST['email'], FILTER_VALIDATE_EMAIL) == false)
echo "not a valid email address!"
```

Dates

`date_create_from_format` way to convert string into date with a specified format

`date_get_last_errors` to get the error if the input doesnt match the template

```
$date = date_create_from_format('j/F/y', $_GET['date']);
$last_errors = date_get_last_errors();
if($last_errors['error_count'] != 0)
    echo "Date is not in the correct format!";
```

- `j` — represents days
- `F` — months
- `y` — years

  you can change the order or use different delimiters

Most types of form data can be validated using regular expressions
preg_match function

```
$pattern = '#^(.+)@([^\.].*)\.([a-z]{2,})$#';
if(preg_match($pattern, $_POST['email']) == false)
    echo "Not a valid email address!";
```

How to create a pattern for regular expression

A *literal* is just a character you wish to match in the target
A *metacharacter* is a special symbol that act as a command to the regular expression parser

- `. [ ] \ ( ) ^ $ | * ? { } +`

Symbols & Syntax

- `^ $` if used at start or end, it means the entire string must match the rest of the expression between `^` and `$` symbols
- `\t` matches a tab character
- `\n` matches a new-line character
- `.` Matches any character other than `\n`
- `[querty]` matches any single character of the set contained within the bracket
- `[^querty]` matches any single character not contained within the brackets
- `[a-z]` matches any single character with range of characters
- `\w` matches any word character (equivalent to `[a-zA-Z0-9]`)
- `\W` matches any non word character
- `\s` matches any white space character
- `\S` matches any non-white space character
- `\d` matches any digit
- `\D` matches any non digit
- `*` indicates zero or more matches
- `+` indicates one or more matches
- `?` indicates zero or one match
- `{n}` indicate exactly n matches
- `{n,}` indicates n or more matches
- `{n,m}` indicates at least n but no more than m matches
- `|` matches any one of the terms separated by |. Equivalent to OR
- `()` groups a subexpression