This lecture

The main purpose of this lecture is to cover HTML forms and how a PHP script can obtain values from the user.

Arrays

One of the most useful and interesting data structures in PHP is the array.

Although PHP has much syntax in common with Java and C, arrays are treated quite differently.

An array can be created using the `array()` construct.

```php
<?php
$a = array(10,20,30);
?>
```

Not unexpectedly this creates an array with three elements: $a[0], $a[1], $a[2]

Array Elements

Just as with regular variables there is no need to specify the type of an array, because each variable can be of different type, even another array.

```php
<?php
$a = array(10,20,30);
$b = array(1,2,"apple");
$c = array(1,2,"apple",array(10,20,30));
?>
```

After these assignments $b[2] is a string, while $c[3] is an array.

To access the individual elements of $c[3], we use the intuitive syntax $c[3][0], $c[3][1], $c[3][2]
Examining arrays

The `print` function is not much use with an array:

```php
print $c;
```

produces the single word `Array` as its output. However PHP has a very helpful function `print_r` which prints each array element so that

```php
print_r $b;
```

produces

```php
Array
(
[0] => 1
[1] => 2
[2] => apple
)
```

Why the strange output?

You may wonder why the output has the form

```php
[0] => 1
[1] => 2
```

rather than just listing the elements in order. The reason for this is that in PHP all arrays are associative arrays — that is, the indices need not be numbers starting from 0, but can be anything. So

```php
$snum = array("smithd01" => 19541023, "jonesb01" => 23210813);
```

creates an array where the elements are called

```php
$snum["smithd01"], $snum["jonesb01"]
```

Array looping

If an array has normal numerical indices, then it is easy to loop through the array using a C/Java style `for` loop.

```php
for ($i = 0; $i < count($b); $i++) {
    print "<p>Element $i has value $b[$i]</p>";
}
```

But how do we process the elements in an associative array? Or the elements in an array where the indices are not consecutive?

To enable these operations, PHP has a special looping construct called `foreach`.

Recursive print

The `print_r` is a mnemonic name for “print recursively” so

```php
print_r $c;
```

yields

```php
Array
(
[0] => 1
[1] => 2
[2] => apple
[3] => Array
(
[0] => 10
[1] => 20
[2] => 30
)
)
```
**Arrays in PHP**

**Associative array looping**

The basic syntax of the `foreach` loop is

```php
foreach ($array as $key => $value) {
    // process the element [$key] => [$value]
}
```

where `$array` is an array expression, and `$key` and `$value` are two variable names.

This loop will examine every entry of the array, in turn setting the two variables `$key` and `$value` to the index/value pair corresponding to the current entry of the array.

(DF Royle 2008-8, N Spadaccini 2008)

**Example**

Assume `$snum` is the array given earlier. Then

```php
foreach ($snum as $email => $num) {
    print "<p>UserID $email has student number $num</p>";
}
```

would print out the details for each array element.

The `foreach` syntax essentially allows the user to get hold of both the index and the value for each

(I have used the term “index” in analogy with normal arrays, but it is more usual to visualize an associative array as consisting of “key/value” pairs.)

(DF Royle 2008-8, N Spadaccini 2008)

**HTML Forms**

We have covered arrays fairly extensively because this is how a PHP script receives the values that are entered into an HTML form.

An HTML form is a document that contains, in addition to normal content, a number of special elements called *controls* which are items such as buttons and various input selectors such checkboxes, radio buttons, menus and textfields.

Users interact with a form through these controls by making menu selections, checking checkboxes or radiobuttons or entering values into textfields.

When the user is satisfied, the form is *submitted* and the names and values of the controls are made available to the webserver.

(DF Royle 2008-8, N Spadaccini 2008)

**Form Structure**

An HTML form is delimited by a `<form>` .. `<form>` pair and all the controls for that particular form must appear between the tags.

```html
<form>
</form>
```

In order for a form to do anything, it must have at least three items:

- The name of a *script* (e.g. a PHP page) that will process the form
- The name of the protocol used to send the control's values to the script
- A mechanism to cause the form to be submitted.

(DF Royle 2008-8, N Spadaccini 2008)
## Action and Method

```html
<form
  method="post">
</form>
```

The value of the `action` attribute is the URL of the *destination script* (in this case `register.php`) that will receive the values of the controls. The value of the `method` attribute is either "get" or "post" and indicates which protocol should be used to send the values to the destination script.

(GF Royle 2006-8, N Spadaccini 2008)

## A submit button

The user will usually press a button labelled “Submit” or “Send” or some suggestive name to cause the form to be submitted.

```html
<form
  method="post">
  <input type="submit" value="Register">
</form>
```

A submit button is added to the form by using the HTML `<input/>` element. The `type` attribute indicates what type of button it is (in this case a submit-button) and the `value` attribute gives the label that will appear on the button.

(GF Royle 2006-8, N Spadaccini 2008)

## A control

One of the simplest types of control is a single-line *textfield* which is just a different type of `<input/>` element.

```html
<form
  method="post">
  <input type="text" name="email">
  <input type="submit" value="Register">
</form>
```

This creates a control of type *text* whose name is `email`.

(GF Royle 2006-8, N Spadaccini 2008)

## What does this look like?

This control now displays a single line textfield for the user to enter some information.

Notice that the `name` of the control (`email`) does not appear anywhere to the user. Assuming we want the user to actually enter their email address here, we would need to inform them somehow.
What happens next?

Now assume the user types in an email address and then presses the submit button.

The destination script register.php is then executed with the values of the form controls available to it in the array $_GET or $_POST depending on whether the form specified the “get” or “post” method.

The register.php script can then extract the value from the array using the name of the control as the key.

```php
<?php
    $user = $_POST['email'];
?>
```

Further processing

The register.php script can now use the supplied email address in various ways – for example, you can use the PHP function mail() to send an email message to the user informing them that their registration has succeeded.

```php
<?php
    $user = $_POST['email'];
    // add to database
    $subject = "Registration Confirm";
    $message = "You have been registered as $user";
    mail($user, $subject, $message);
?>
```

Initial values

The programmer can specify an initial value for the control; this might be a sensible default value or even instructions.

```html
<input type="text" name="email" value="Type your email here"/>
```

Controls

There are a variety of other <INPUT> types that can be used in forms.

- **text** for a single line textfield
- **password** for a single line password field
- **radio** for radio buttons
- **checkbox** for checkboxes
- **reset** for a reset button
- **hidden** for a hidden field
Radio buttons

Radio buttons are used when you have a collection of mutually exclusive choices.

For example, an online lunch bar company might allow you to choose between three types of bread for your sandwich.

```html
<input type="radio" name="bread" value="white" checked>
<input type="radio" name="bread" value="brown">
<input type="radio" name="bread" value="multi">
```

Notice that there are three input elements *all with the same name* (in this case `bread`). The browser will ensure that only one of these input elements can be selected. The `checked` attribute indicates that the element with value `white` is selected by default.

At the destination

The destination script can then access the variable `$_POST['bread']` or `$_GET['bread']` to determine which type of bread was selected.

Add the text

```html
<h3>Choose your bread</h3>
<p>
White: <input type="radio" name="bread" value="white" checked>
Brown: <input type="radio" name="bread" value="brown">
Multigrain: <input type="radio" name="bread" value="multi">
</p>
```
Checkboxes

Checkboxes are input elements used when more than one choice can be made at a time.

Lettuce: <input type="checkbox" value="lettuce" checked>
Carrot: <input type="checkbox" value="carrot" checked>
Tomato: <input type="checkbox" value="tomato">
Alfalfa: <input type="checkbox" value="alfalfa">

What should the name be?

The checkbox group does not have a name yet – we could of course give each item a separate name and then check for them individually. But the code is much cleaner if we give the entire group an array name—signified by appending [] to the name.

Lettuce: <input type="checkbox" name="filling[]" value="lettuce" checked>
Carrot: <input type="checkbox" name="filling[]" value="carrot">
Tomato: <input type="checkbox" name="filling[]" value="tomato" checked>
Alfalfa: <input type="checkbox" name="filling[]" value="alfalfa">

What does the script get?

In this case, if the default values were accepted by the user then the script would receive an array

$_POST['filling']

such that

$_POST['filling'][0] = "lettuce"
$_POST['filling'][1] = "tomato"

What happens if the user selects none of the options?

Nonexistent Variables

If the user selects none of the salad fillings, then there is no name/value pair submitted for that control. This means that the PHP script must be able to check if a variable exists or not before attempting to use it.

This can be done with the function isset()

```php
if (isset($_POST['filling'])) {
    ...
} else {
    print "No salads\n";
}
```