Web Forms

Introduction

- Once an HTTP request arrives, the server decodes the data.
- If the request is for a PHP script, the server passes it on to the PHP engine.
- When the server sends back its reply, it first writes a set of response headers.
- The response headers include important information for the client (e.g. content type being returned etc.).

HTML Forms

- In most cases, the PHP script will interact with the end users clients using one of the two HTTP methods: GET & POST.
- The primary different between GET & POST, is the way through which the additional data is sent by the client.

  When using GET the additional data is sent as a query string, and therefore its amount is limiedt. When using POST the additional data is sent as part of the request, and for that reason its amount is not limited.

HTML Forms

```
<FORM ACTION=account.php METHOD=GET>
    Name: <INPUT TYPE=text name='name'>
    ID: <INPUT TYPE=text name='id'>
    <INPUT TYPE=SUBMIT>
</FORM>
```

HTML Forms

- When a form is submitted using GET, its values are encoded in the query string portion of the URL.

  `http://www.zindell.com/page.php?list=username&id=1212`

- When a form is submitted using POST, its values are part of the request.

The `$_GET` Super Global Array

- Each argument sent with the request (using GET method) is accessible through the `$_GET` super global array.
- The key for each argument is the name of that argument.

The `$_GET` Super Global Array

```
<FORM ACTION=rectangle.php METHOD=GET>
   Width <INPUT TYPE=TEXT NAME=width>
   Height <INPUT TYPE=TEXT NAME=height>
   <INPUT TYPE=SUBMIT>
</FORM>
```

The $_GET Super Global Array

```
<?PHP
   echo $_GET['width'];
   echo "<BR>";
   echo $_GET['height'];
?>
```

The `$_GET` Super Global Array

- Sending the PHP script parameters that include the array notation (brackets signs within their name) will create arrays.

  `http://jacado.com/smpl.php?id=12&book[author]=haim&book[title]=java`

  We can now write a PHP script that access these variables the following way:

```
echo $_GET['book']['author'];
echo $_GET['book']['title'];
```

The `urlencode()` Function

- When sending data via the query string there are specific characters we must exclude.
- Calling the `urlencode()` function takes away all problematic characters, replacing them with substitutes.

```
string urlencode (string $str)
```

This function encodes a given string and makes it feasible to include it within a query string as part of a URL address.

The `$_POST` Super Global Array

- Each argument sent with the request (using POST method) is accessible through the `$_POST` super global array.
- The key for each argument is the name of that argument.
- As with GET, when using POST we can again use the array notation.

The `$_REQUEST` Super Global Array

- When there is a need to write a script that works both with GET and POST, we can use the `$_REQUEST` super global array.

Uploading Files

- Uploading files can be done through a “multi-part” HTTP POST transaction.

```
<FORM ENCTYPE=”multipart/form-data” ACTION=”uploader.php” METHOD=”post”>
    <INPUT TYPE=”hidden” NAME=”MAX_FILE_SIZE” value=”1000”/>
    <INPUT TYPE=”file” NAME=”file_data”/>
    <INPUT TYPE=”submit” VALUE=”send file”/>
</FORM>
```

- Once the file upload completed, the file is stored in a temporary location. It is our PHP script responsibility to copy it to another location.

The `$_FILES` Super Global Array

- Each element of this array includes a key which is the name of the HTML element that uploaded a file.
- Each element of this array includes a value which is an array with the following elements:

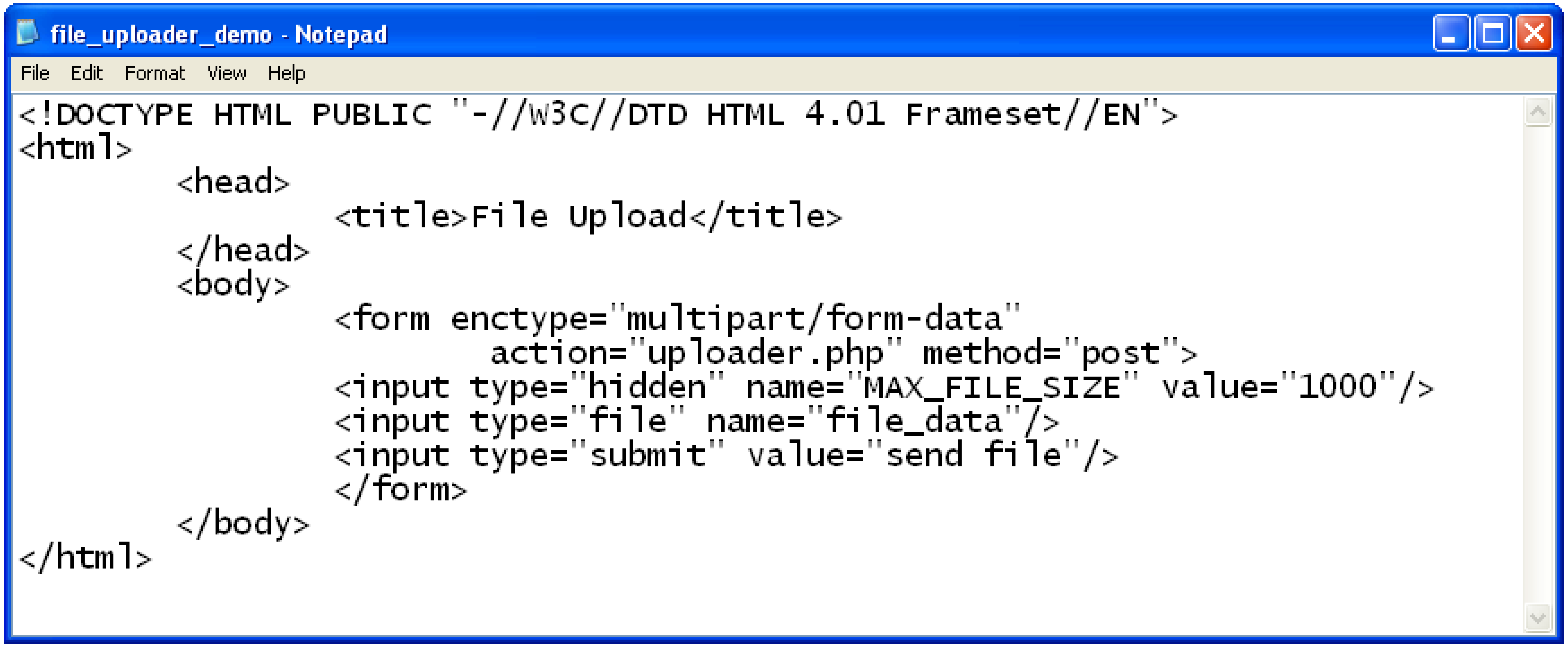
name	The original file name
type	The MIME type
size	The file size
tmp_name	The name of the file's temporary location
error	The error code

Uploading File Demo

- The following demo shows how a file is uploaded on the server and how the PHP script saves it with a new name.

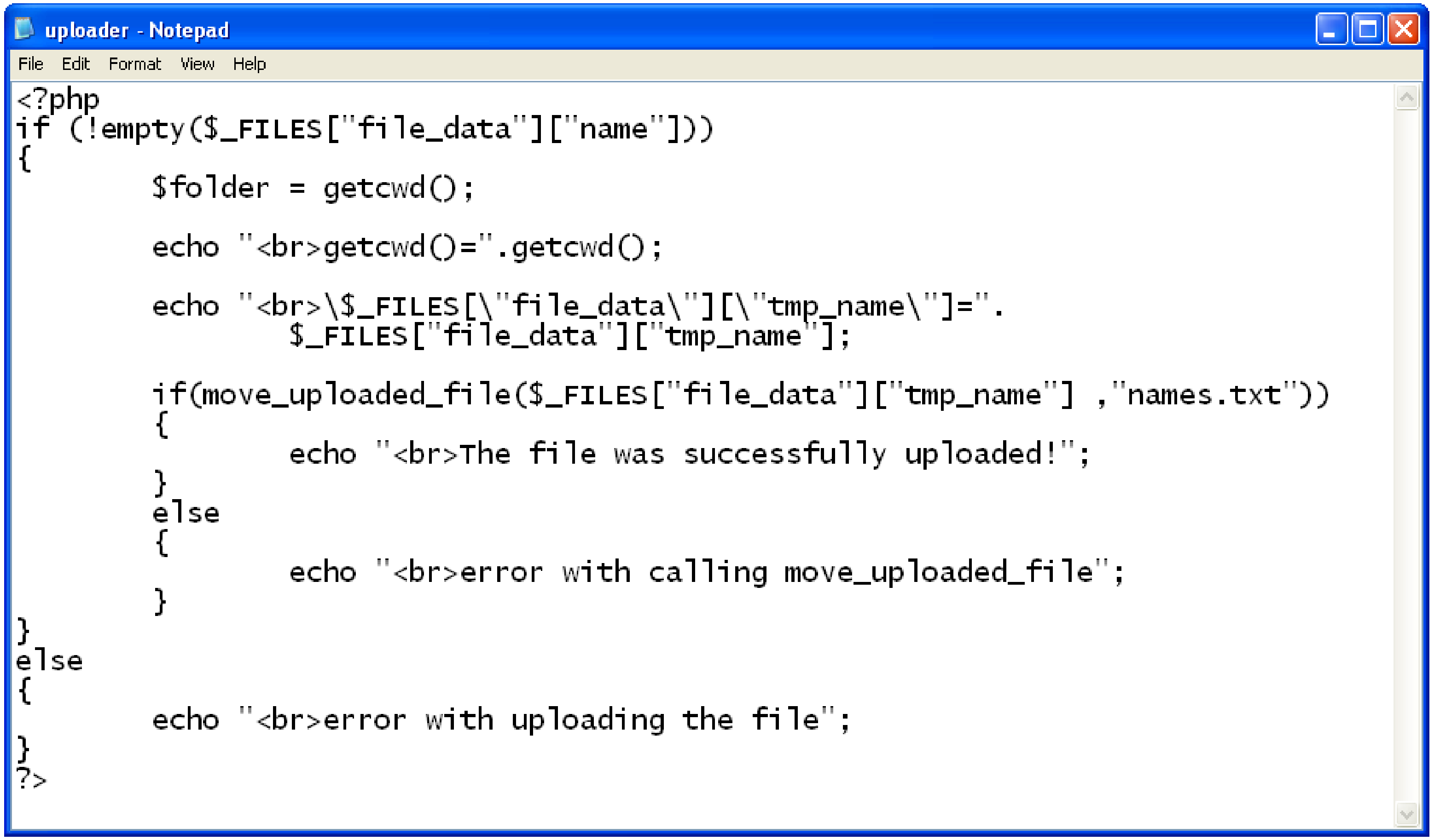
  You can download the files from the samples folder and you can find a short video clip that shows how does it work.

Uploading File Demo



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
        <head>
                <title>File Upload</title>
        </head>
        <body>
                <form enctype="multipart/form-data"
                        action="uploader.php" method="post">
                <input type="hidden" name="MAX_FILE_SIZE" value="1000"/>
                <input type="file" name="file_data"/>
                <input type="submit" value="send file"/>
                </form>
        </body>
</html>
```

Uploading File Demo



```
<?php
if (!empty($_FILES["file_data"]["name"]))
{
	$folder = getcwd();

	echo "<br>getcwd()=".getcwd();

	echo "<br>\$_FILES[\"file_data\"][\"tmp_name\"]=".
		$_FILES["file_data"]["tmp_name"];

	if(move_uploaded_file($_FILES["file_data"]["tmp_name"] ,"names.txt"))
	{
		echo "<br>The file was successfully uploaded!";
	}
	else
	{
		echo "<br>error with calling move_uploaded_file";
	}
}
else
{
	echo "<br>error with uploading the file";
}
?>
```

Input Validation

- In order to cope successfully with hacker who try to harm our web application we better validate the data our PHP script receives.
- We cannot count on client side validation using Java Script. Hackers can easily remove it. Client side validation using JavaScript contributes to our web application usability. It doesn't contribute to its security.

Input Validation

- It is a common practice to define variables we can easily identify as variables that hold validated clean values. Including the word 'clean' or a similar one in the name of each and every clean variable will differentiate them from all others.

```
$email_address = $_GET['email'];
//validating email address
$email_address_clean = ...
```

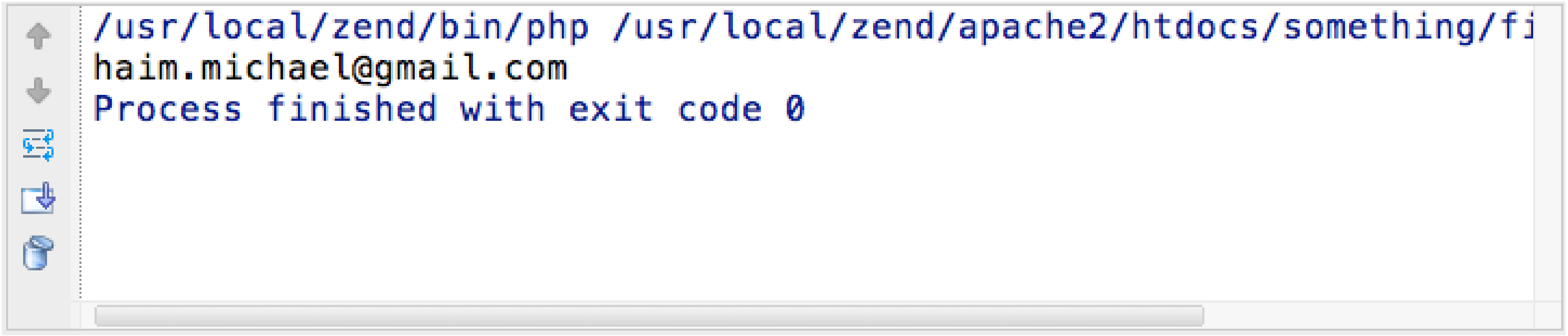
Input Validation

- The function `filter_var` that exists in PHP as of version 5.2 provides us with the simplest way to validate our input.

Input Validation

```
<?php
$email_address_dirty = "haim.michael@gmail.com";
if(filter_var($email_address_dirty,FILTER_VALIDATE_EMAIL))
{
    $email_address_clean = $email_address_dirty;
}
echo $email_address_clean;
?>
```

Input Validation



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/fi
haim.michael@gmail.com
Process finished with exit code 0
```

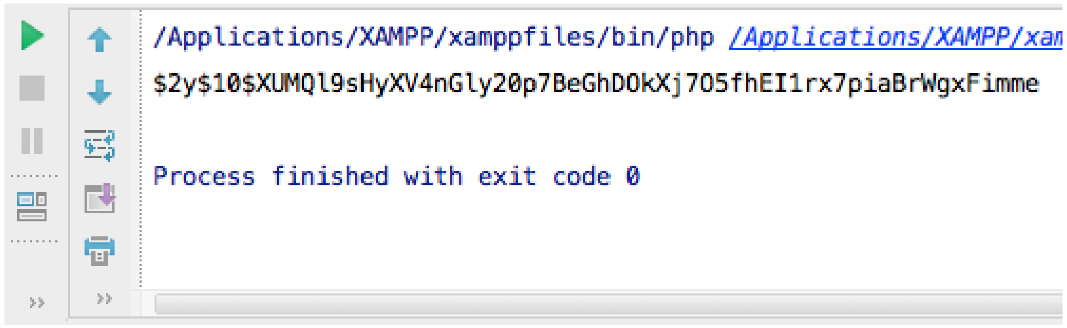
The `password_hash` Function

- This function receives the password that needs to be hashed and returns its hash value.
- The first argument is the password we need to hash. The second argument is an integer value that represents the specific algorithm we want to use.

The `password_hash` Function

```
<?php
$str = "haimmichael";
echo password_hash($str, PASSWORD_DEFAULT)."\n";
?>
```

The `password_hash` Function



```
/Applications/XAMPP/xamppfiles/bin/php /Applications/XAMPP/xam
$2y$10$XUMQl9sHyXV4nGly20p7BeGhDOkXj7O5fhEI1rx7piaBrWgxFimme

Process finished with exit code 0
```

The Output

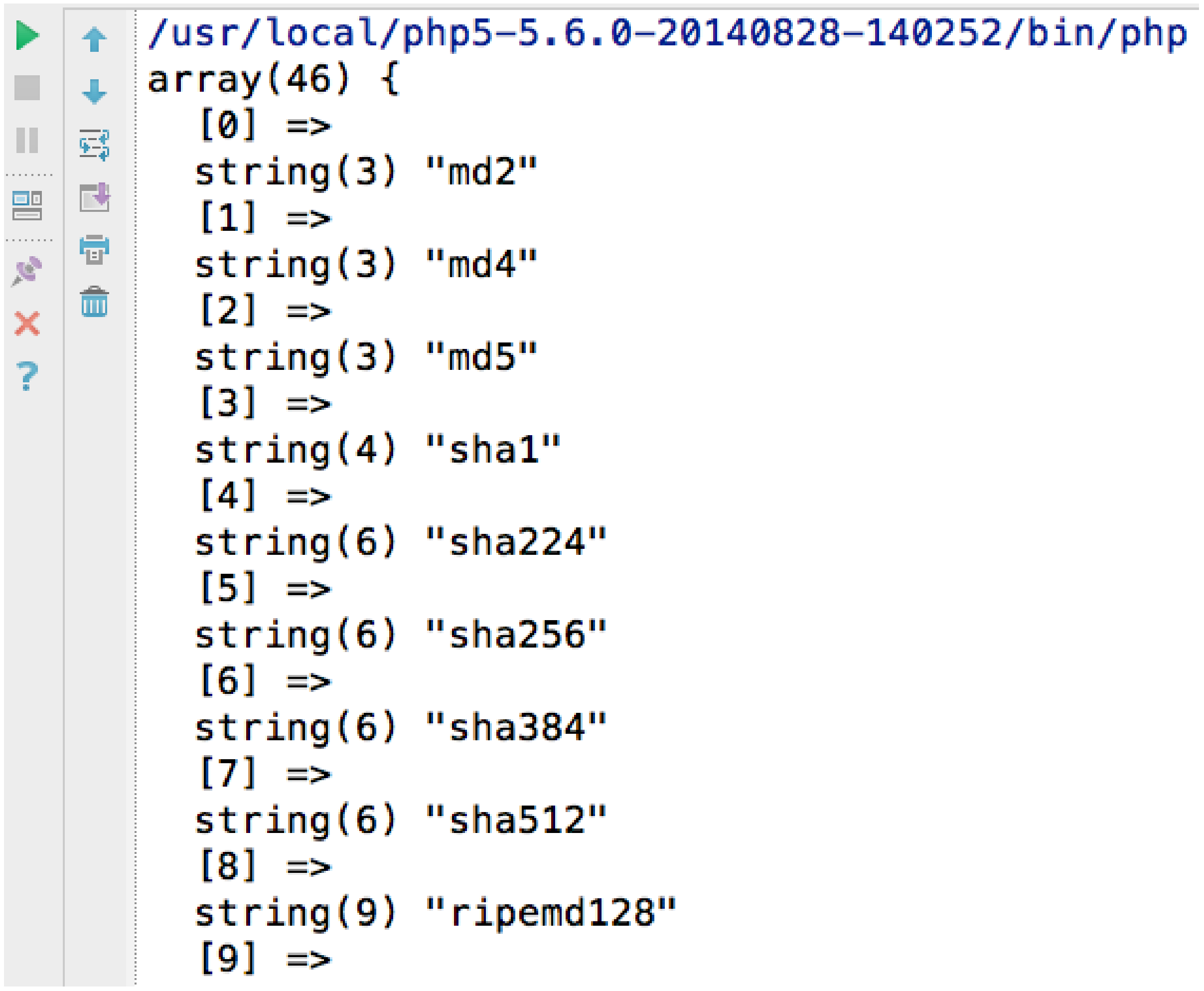
The `hash` Function

- This function calculates the hash value for the data we have. Unlike the `password_hash` function, using the `hash` function we have more algorithms to choose from. PHP 5.6 adds the support for the `gost-crypto` algorithm.

The `hash` Function

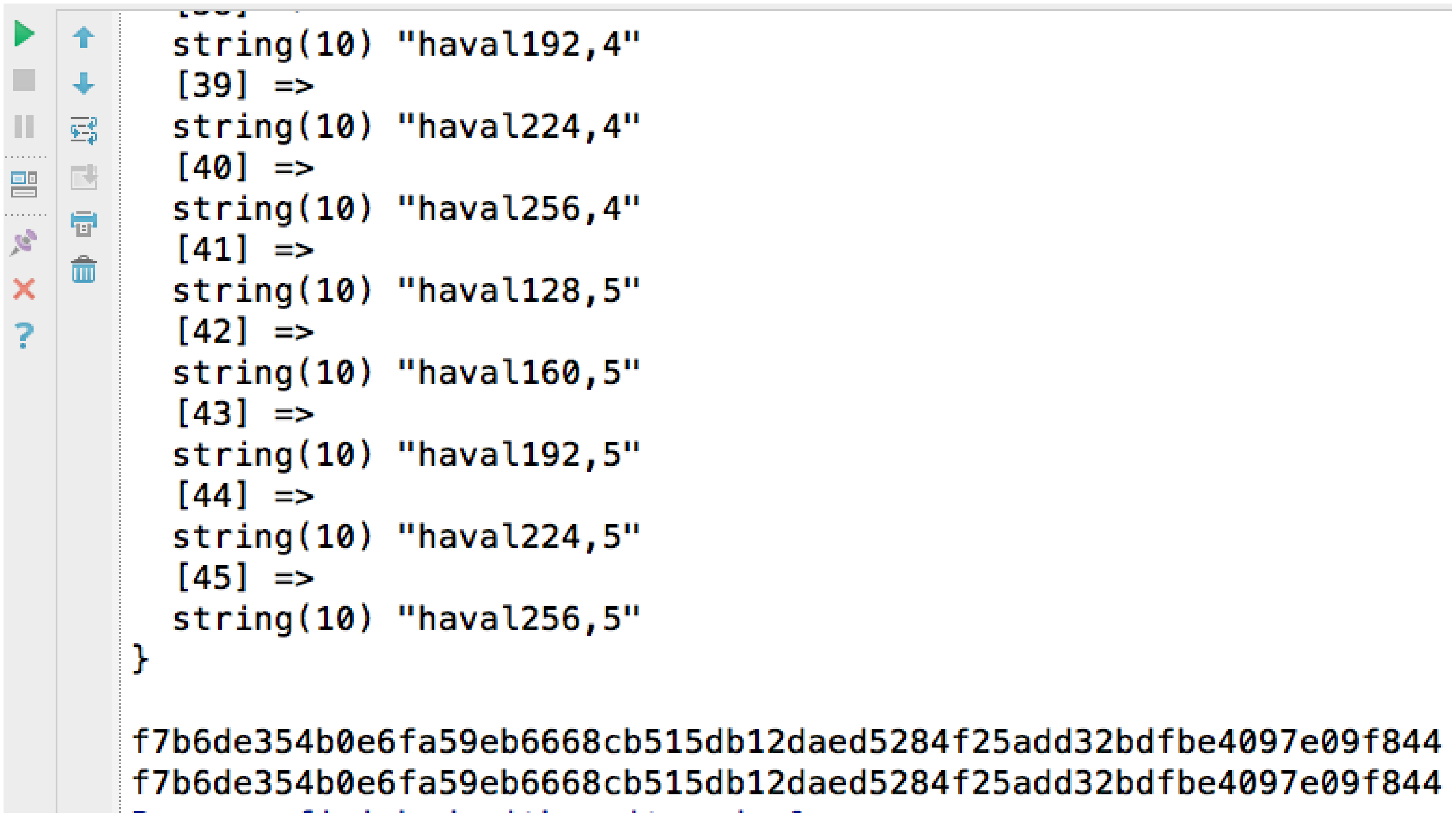
```
<?php
$vec = hash_algos();
var_dump($vec);

$password = "abcjojo123";
echo "\n".hash("gost-crypto",$password);
echo "\n".hash("gost-crypto",$password);
?>
```

The `hash` Function



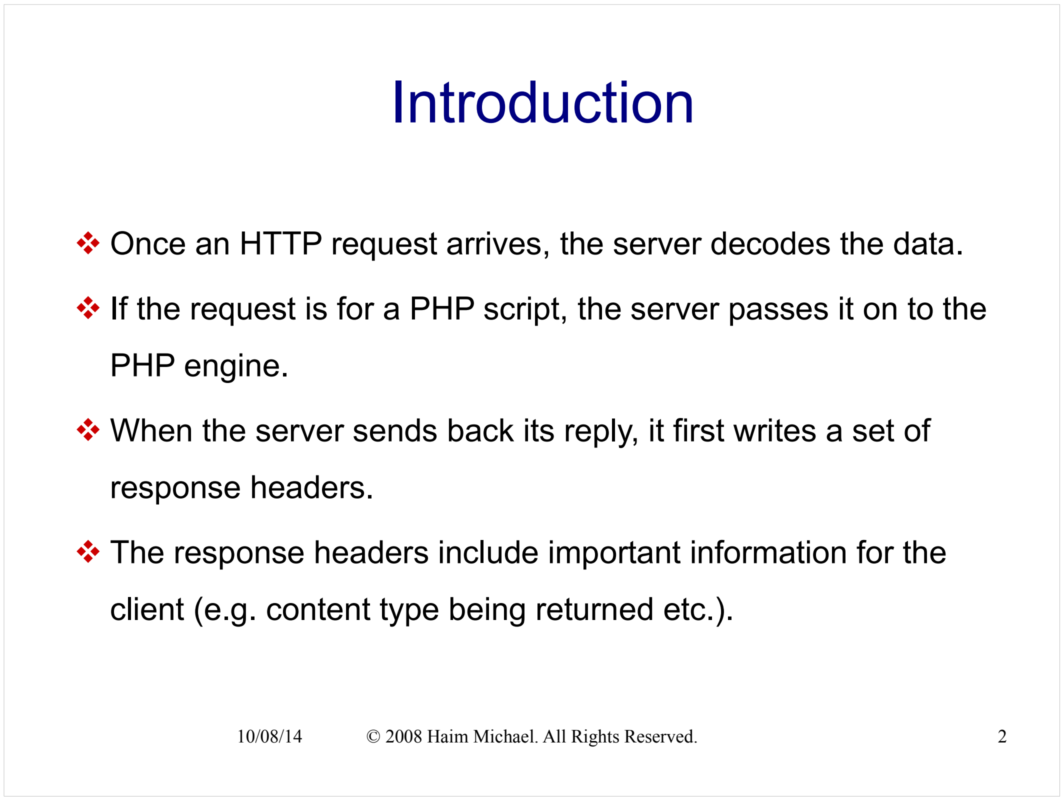
```
/usr/local/php5-5.6.0-20140828-140252/bin/php
array(46) {
  [0] =>
  string(3) "md2"
  [1] =>
  string(3) "md4"
  [2] =>
  string(3) "md5"
  [3] =>
  string(4) "sha1"
  [4] =>
  string(6) "sha224"
  [5] =>
  string(6) "sha256"
  [6] =>
  string(6) "sha384"
  [7] =>
  string(6) "sha512"
  [8] =>
  string(9) "ripemd128"
  [9] =>
```

The `hash` Function



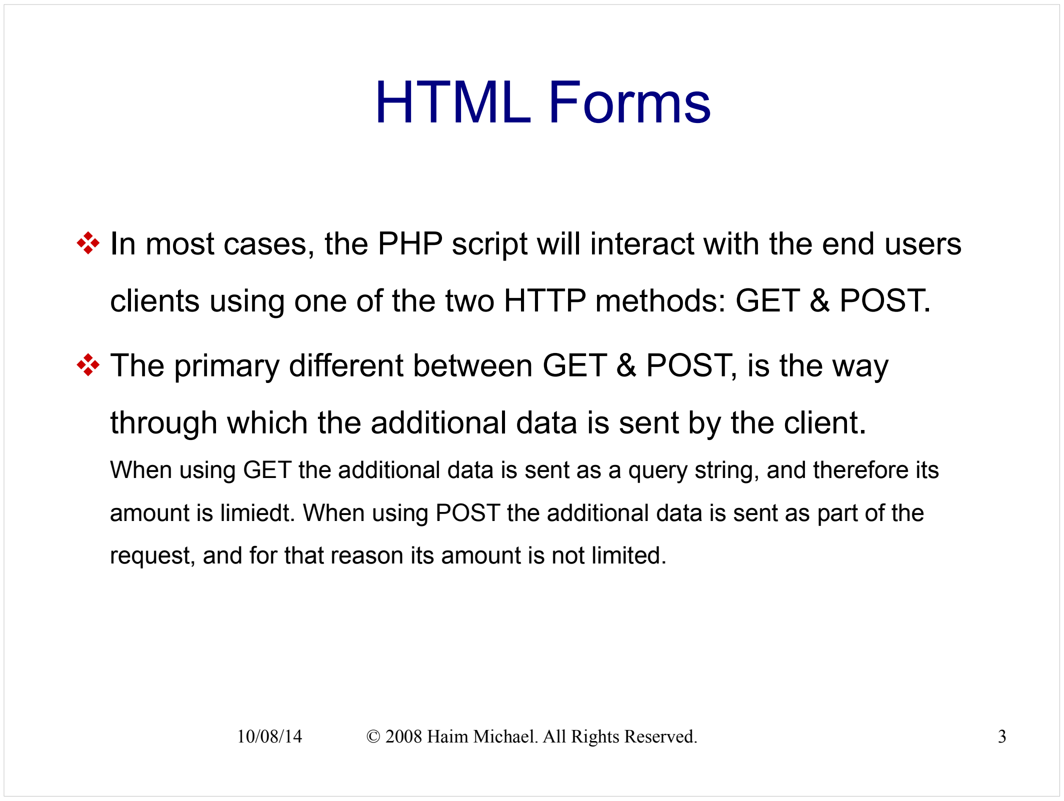
```
  string(10) "haval192,4"
  [39] =>
  string(10) "haval224,4"
  [40] =>
  string(10) "haval256,4"
  [41] =>
  string(10) "haval128,5"
  [42] =>
  string(10) "haval160,5"
  [43] =>
  string(10) "haval192,5"
  [44] =>
  string(10) "haval224,5"
  [45] =>
  string(10) "haval256,5"
}

f7b6de354b0e6fa59eb6668cb515db12daed5284f25add32bdfbe4097e09f844
f7b6de354b0e6fa59eb6668cb515db12daed5284f25add32bdfbe4097e09f844
```

Web Forms



Introduction

- Once an HTTP request arrives, the server decodes the data.
- If the request is for a PHP script, the server passes it on to the PHP engine.
- When the server sends back its reply, it first writes a set of response headers.
- The response headers include important information for the client (e.g. content type being returned etc.).



HTML Forms

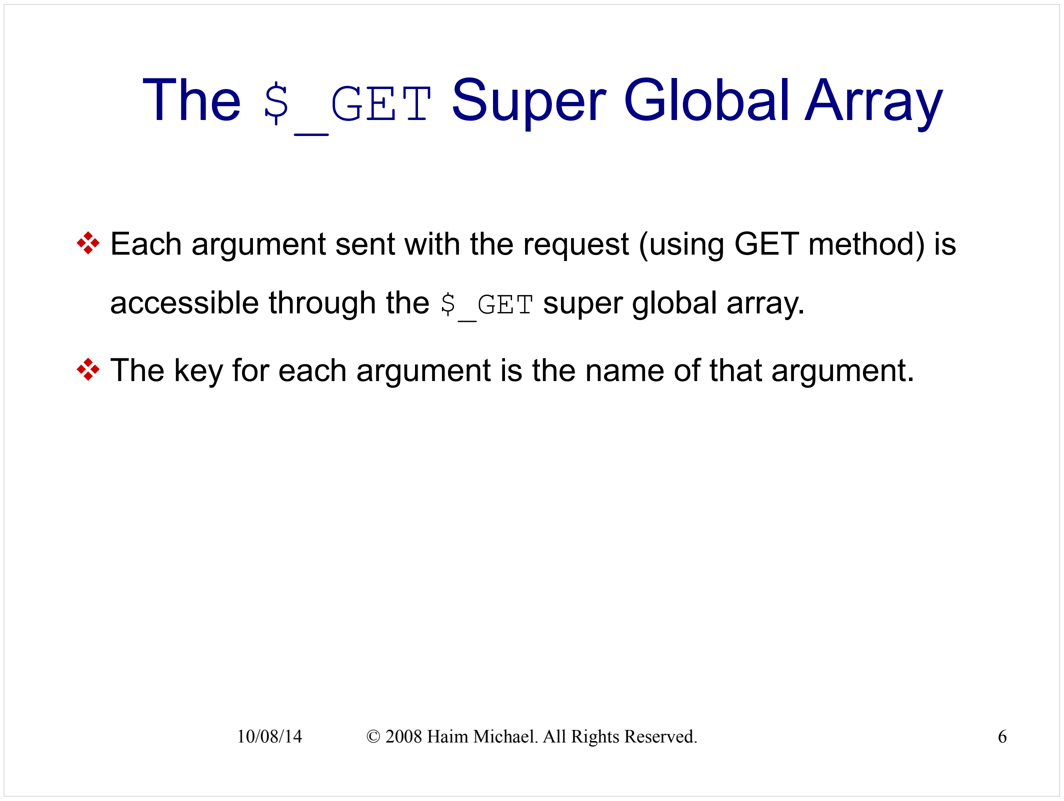
- In most cases, the PHP script will interact with the end users clients using one of the two HTTP methods: GET & POST.
- The primary different between GET & POST, is the way through which the additional data is sent by the client.
  When using GET the additional data is sent as a query string, and therefore its amount is limiedt. When using POST the additional data is sent as part of the request, and for that reason its amount is not limited.

HTML Forms

```
<FORM ACTION=account.php METHOD=GET>
    Name: <INPUT TYPE=text name='name'>
    ID: <INPUT TYPE=text name='id'>
    <INPUT TYPE=SUBMIT>
</FORM>
```

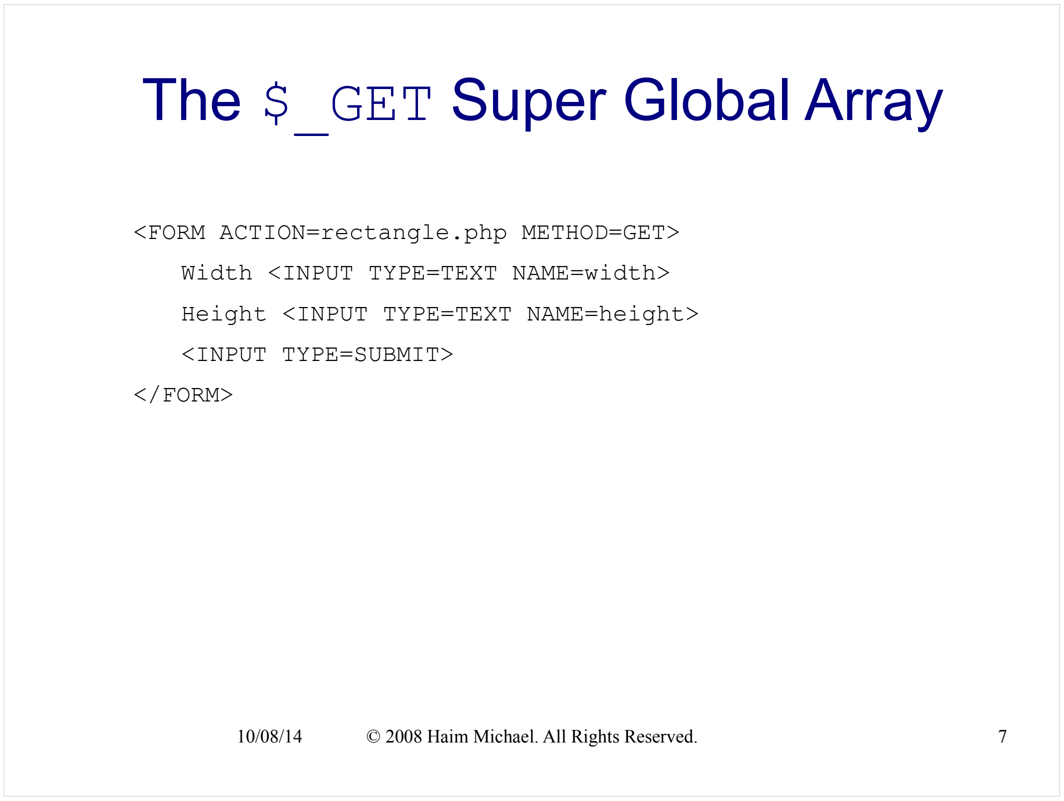
HTML Forms

- When a form is submitted using GET, its values are encoded in the query string portion of the URL.
  `http://www.zindell.com/page.php?list=username&id=1212`
- When a form is submitted using POST, its values are part of the request.



The `$_GET` Super Global Array

- ❖ Each argument sent with the request (using GET method) is accessible through the `$_GET` super global array.
- ❖ The key for each argument is the name of that argument.



The $_GET Super Global Array

```
<FORM ACTION=rectangle.php METHOD=GET>
   Width <INPUT TYPE=TEXT NAME=width>
   Height <INPUT TYPE=TEXT NAME=height>
   <INPUT TYPE=SUBMIT>
</FORM>
```

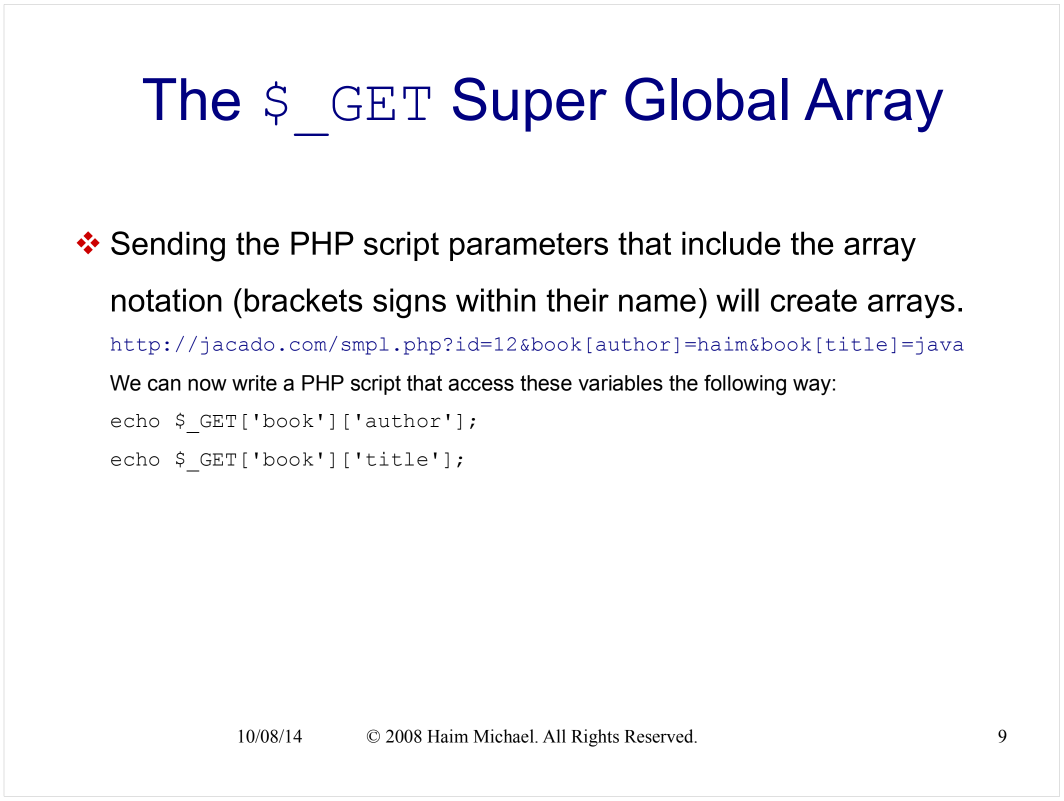
10/08/14 © 2008 Haim Michael. All Rights Reserved. 7

You can find these two sample files in the samples folder of this topic:
rectangle_form.html
rectangle.php

You can execute this sample browsing at
http://www.abelski.com/courses/php/samples/webform/rectangle_form.html

The `$_GET` Super Global Array

```
<?PHP
   echo $_GET['width'];
   echo "<BR>";
   echo $_GET['height'];
?>
```



The `$_GET` Super Global Array

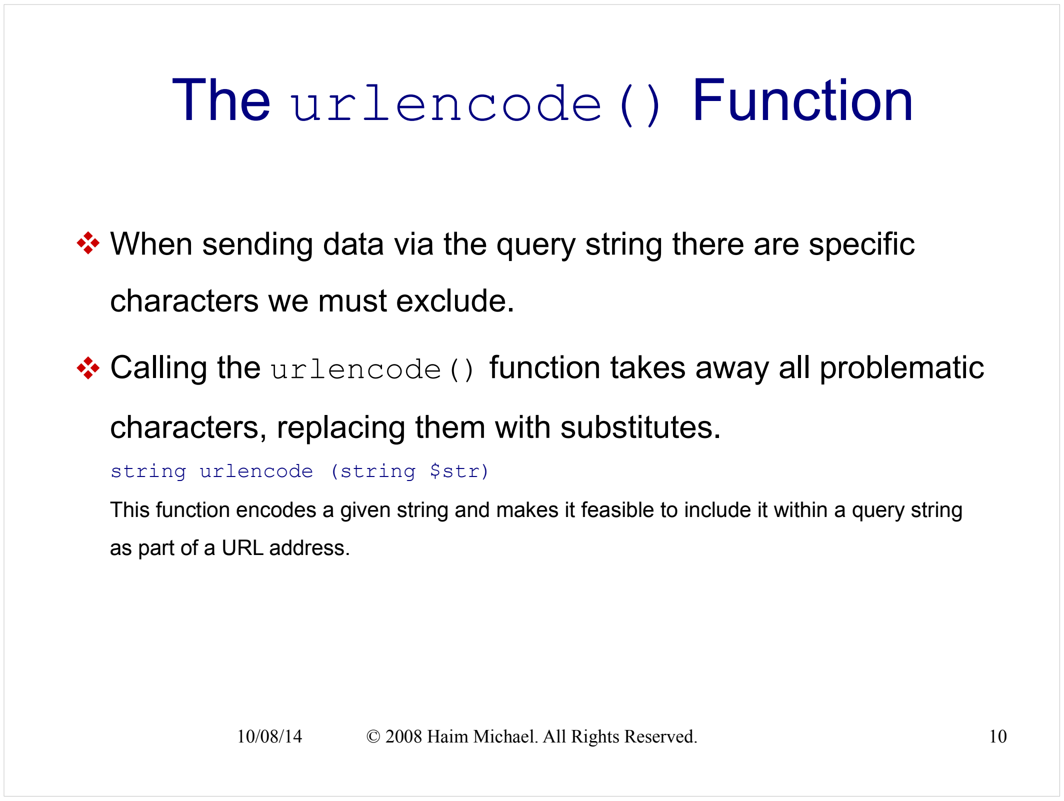
- Sending the PHP script parameters that include the array notation (brackets signs within their name) will create arrays.

  `http://jacado.com/smpl.php?id=12&book[author]=haim&book[title]=java`

  We can now write a PHP script that access these variables the following way:

```
echo $_GET['book']['author'];
echo $_GET['book']['title'];
```

10/08/14 © 2008 Haim Michael. All Rights Reserved. 9



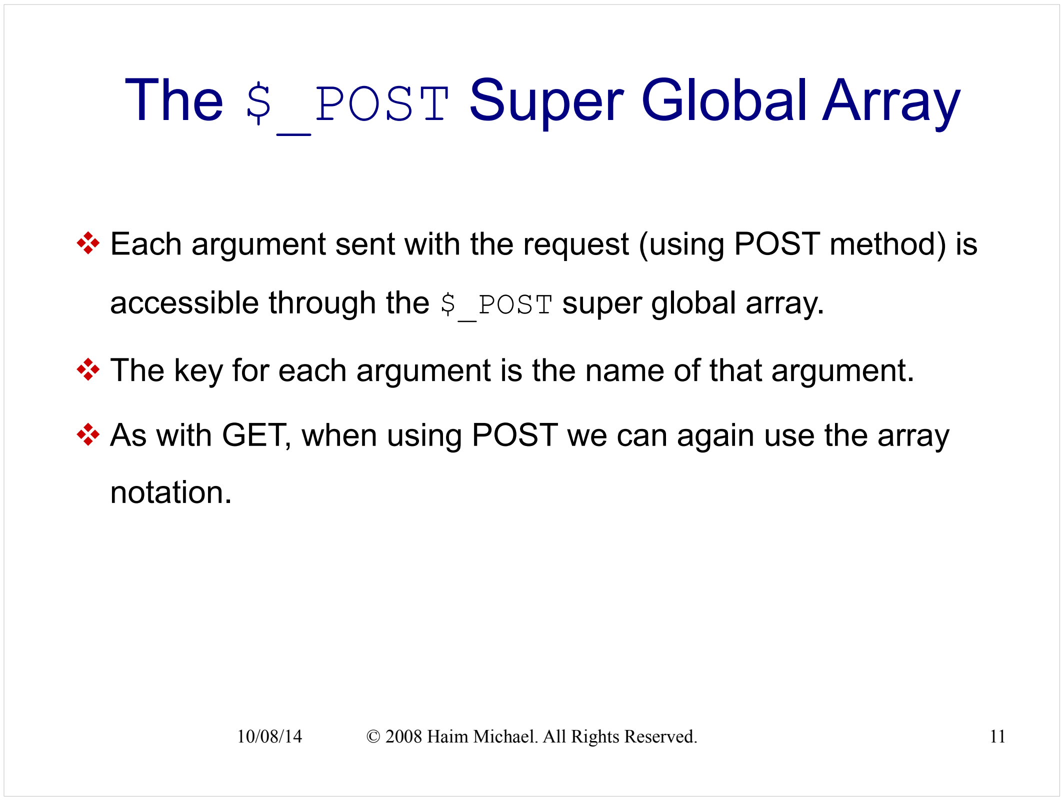
The `urlencode()` Function

- When sending data via the query string there are specific characters we must exclude.
- Calling the `urlencode()` function takes away all problematic characters, replacing them with substitutes.

```
string urlencode (string $str)
```

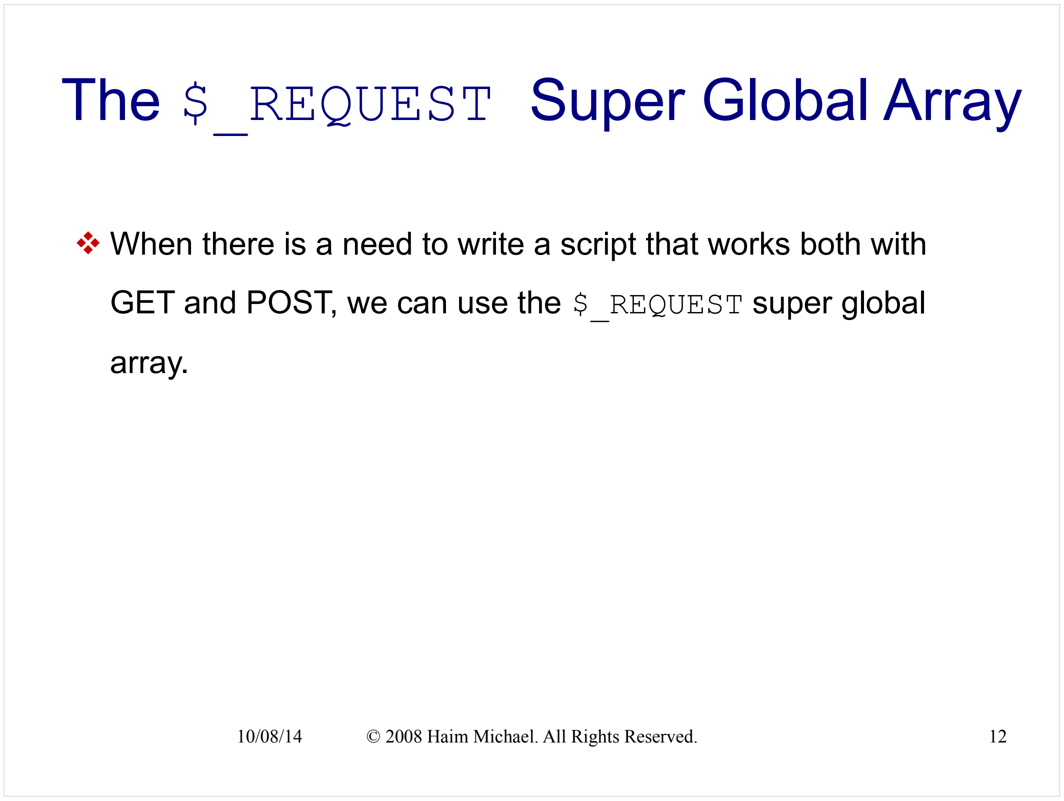
This function encodes a given string and makes it feasible to include it within a query string as part of a URL address.

10/08/14 © 2008 Haim Michael. All Rights Reserved. 10



The `$_POST` Super Global Array

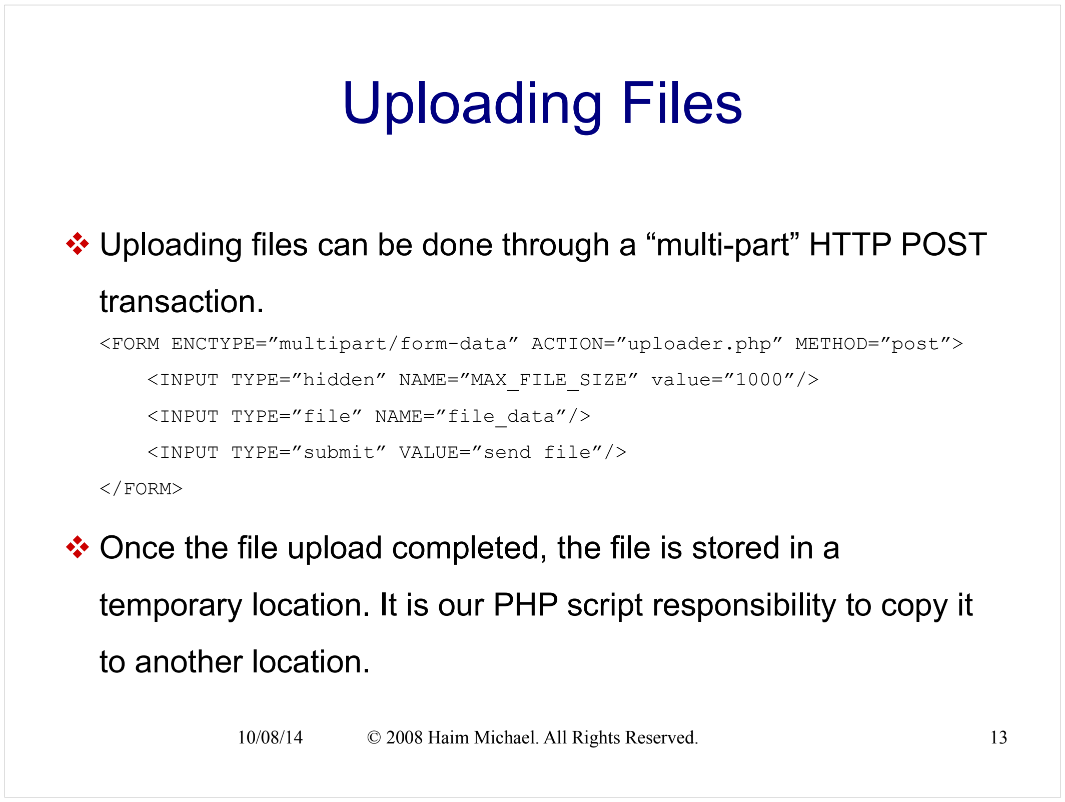
- ❖ Each argument sent with the request (using POST method) is accessible through the `$_POST` super global array.
- ❖ The key for each argument is the name of that argument.
- ❖ As with GET, when using POST we can again use the array notation.



The `$_REQUEST` Super Global Array

❖ When there is a need to write a script that works both with GET and POST, we can use the `$_REQUEST` super global array.

Using the $_REQUEST super global array has a potentially major security issue. This will be covered in the security topic.



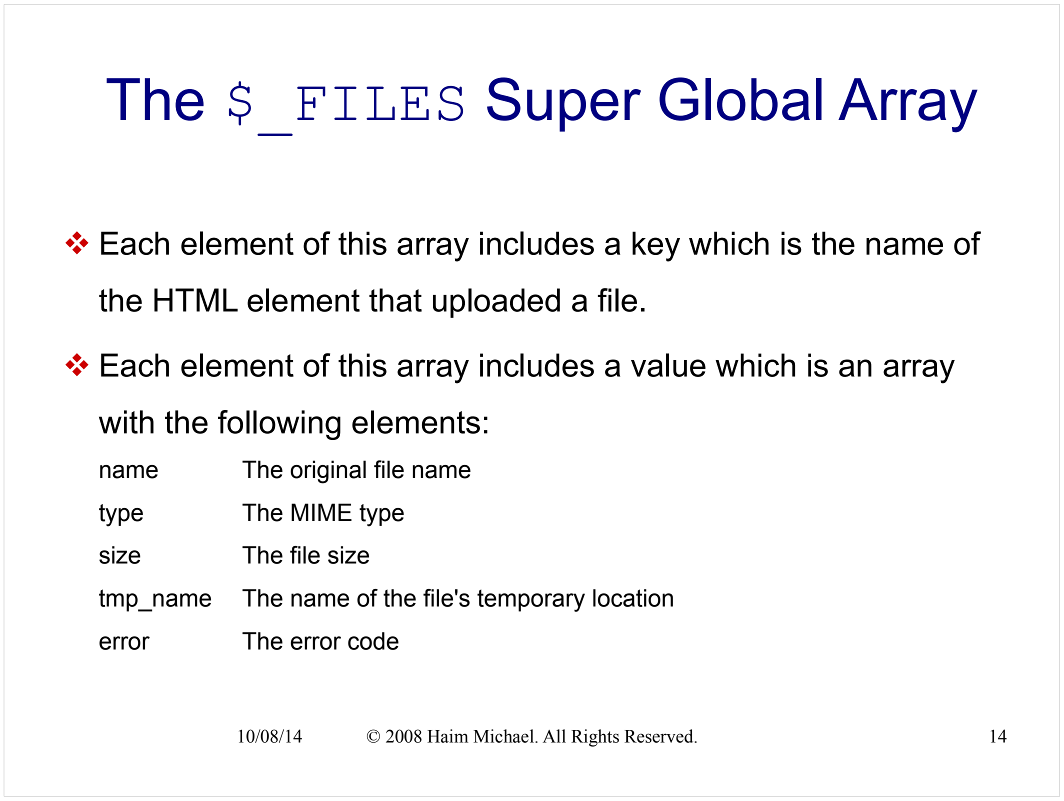
Uploading Files

- Uploading files can be done through a “multi-part” HTTP POST transaction.

```
<FORM ENCTYPE=”multipart/form-data” ACTION=”uploader.php” METHOD=”post”>
    <INPUT TYPE=”hidden” NAME=”MAX_FILE_SIZE” value=”1000”/>
    <INPUT TYPE=”file” NAME=”file_data”/>
    <INPUT TYPE=”submit” VALUE=”send file”/>
</FORM>
```

- Once the file upload completed, the file is stored in a temporary location. It is our PHP script responsibility to copy it to another location.

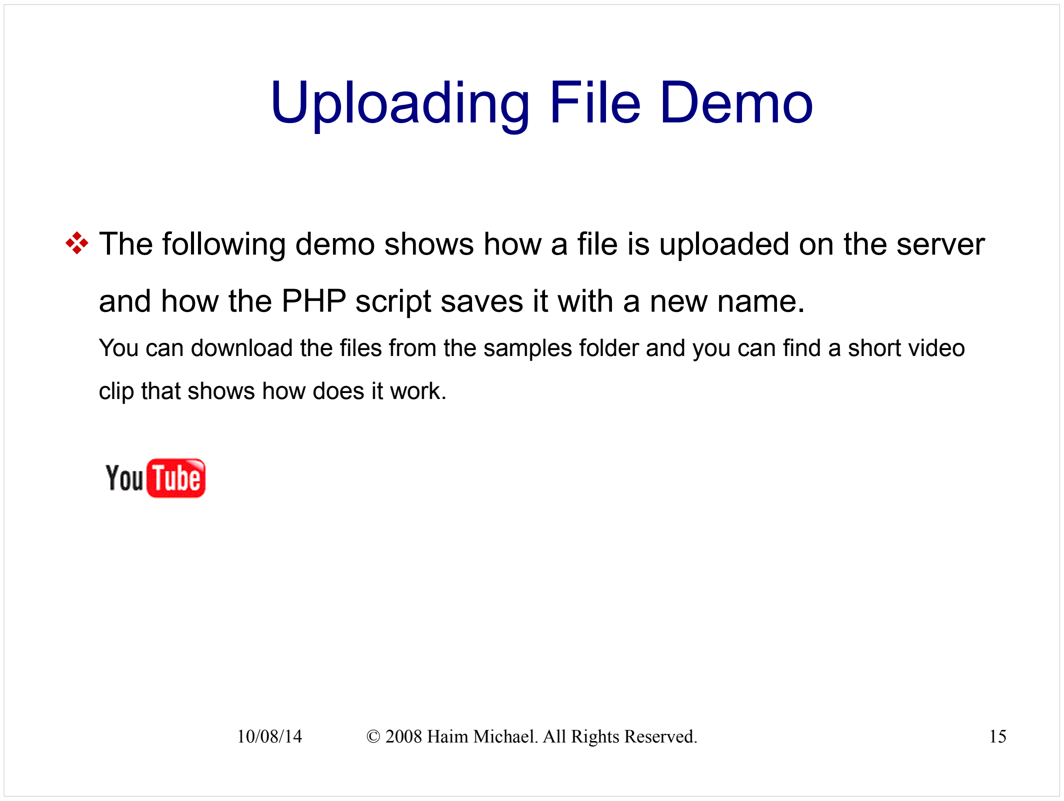
The temporary file is automatically destroyed when the script ends.



The `$_FILES` Super Global Array

- Each element of this array includes a key which is the name of the HTML element that uploaded a file.
- Each element of this array includes a value which is an array with the following elements:

name	The original file name
type	The MIME type
size	The file size
tmp_name	The name of the file's temporary location
error	The error code

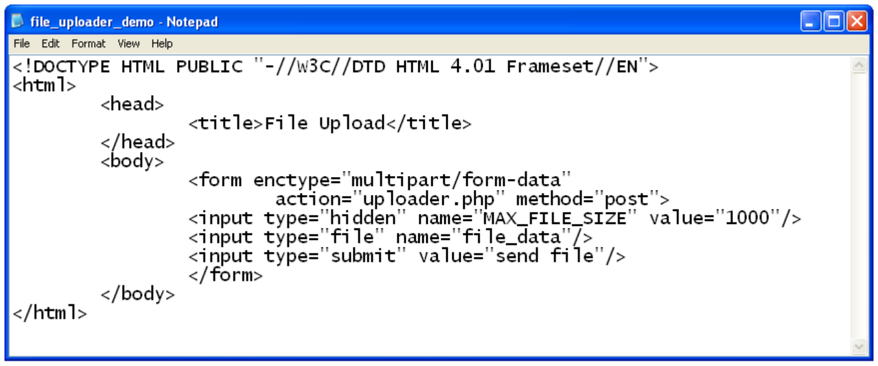


Uploading File Demo

- The following demo shows how a file is uploaded on the server and how the PHP script saves it with a new name.
  You can download the files from the samples folder and you can find a short video clip that shows how does it work.

The temporary file is automatically destroyed when the script ends.

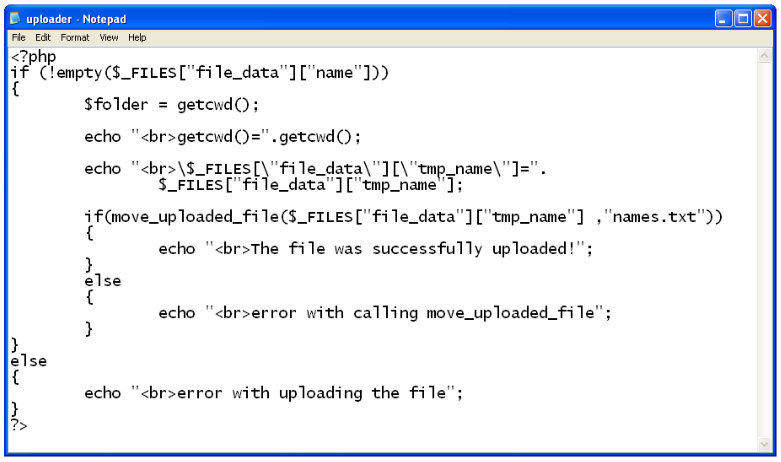
Uploading File Demo



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
        <head>
                <title>File Upload</title>
        </head>
        <body>
                <form enctype="multipart/form-data"
                        action="uploader.php" method="post">
                <input type="hidden" name="MAX_FILE_SIZE" value="1000"/>
                <input type="file" name="file_data"/>
                <input type="submit" value="send file"/>
                </form>
        </body>
</html>
```

The temporary file is automatically destroyed when the script ends.

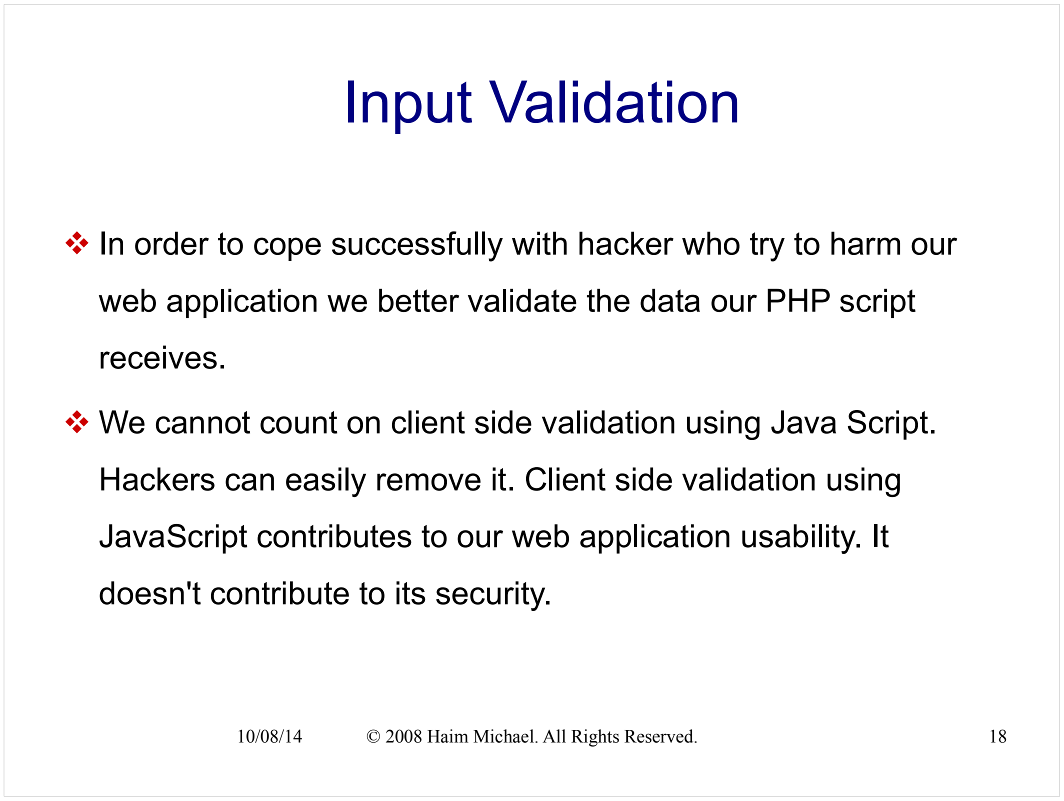
Uploading File Demo



```
<?php
if (!empty($_FILES["file_data"]["name"]))
{
        $folder = getcwd();

        echo "<br>getcwd()=".getcwd();

        echo "<br>\$_FILES[\"file_data\"][\"tmp_name\"]=".
                $_FILES["file_data"]["tmp_name"];

        if(move_uploaded_file($_FILES["file_data"]["tmp_name"] ,"names.txt"))
        {
                echo "<br>The file was successfully uploaded!";

        }
        else
        {
                echo "<br>error with calling move_uploaded_file";
        }
}
else
{
        echo "<br>error with uploading the file";
}
?>
```

10/08/14 © 2008 Haim Michael. All Rights Reserved. 17

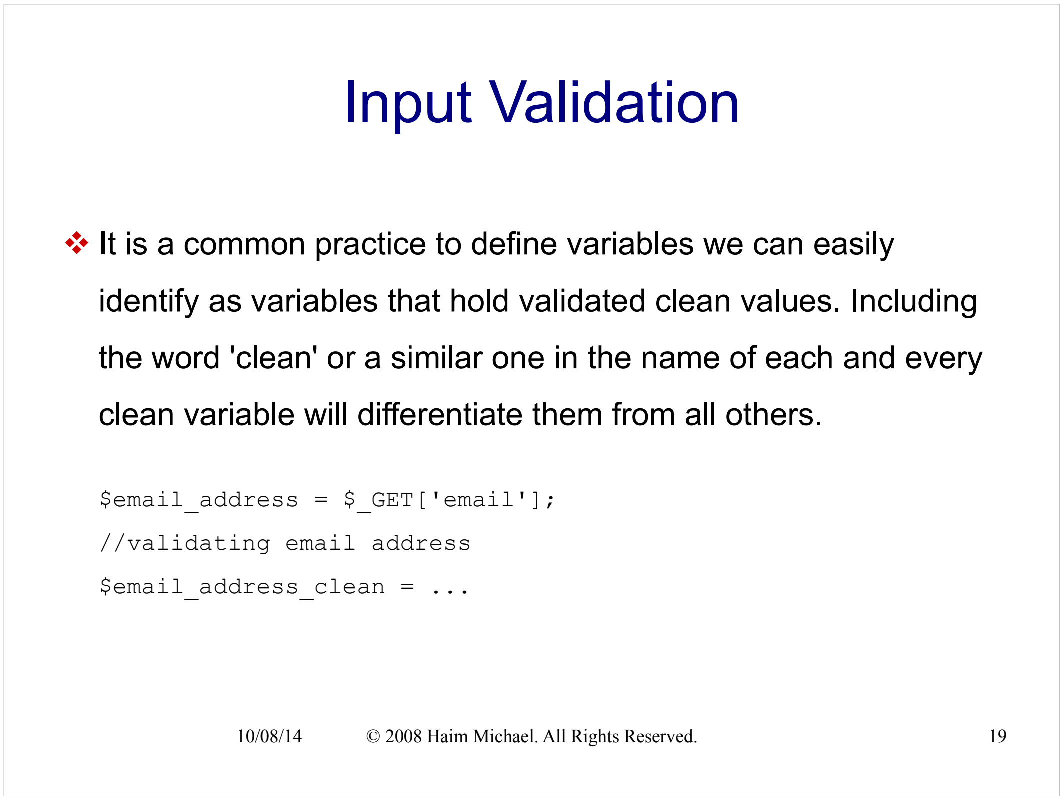
The temporary file is automatically destroyed when the script ends.



Input Validation

- In order to cope successfully with hacker who try to harm our web application we better validate the data our PHP script receives.
- We cannot count on client side validation using Java Script. Hackers can easily remove it. Client side validation using JavaScript contributes to our web application usability. It doesn't contribute to its security.

The temporary file is automatically destroyed when the script ends.



Input Validation

- It is a common practice to define variables we can easily identify as variables that hold validated clean values. Including the word 'clean' or a similar one in the name of each and every clean variable will differentiate them from all others.

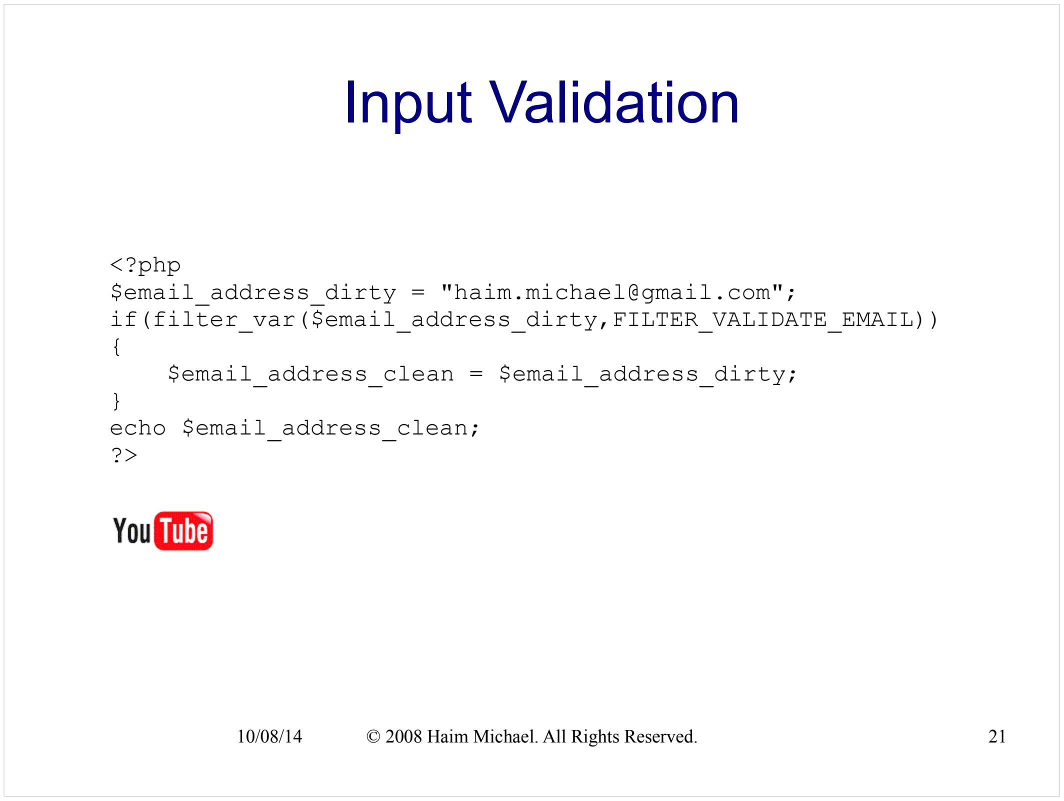
```
$email_address = $_GET['email'];
//validating email address
$email_address_clean = ...
```

The temporary file is automatically destroyed when the script ends.

Input Validation

- The function `filter_var` that exists in PHP as of version 5.2 provides us with the simplest way to validate our input.

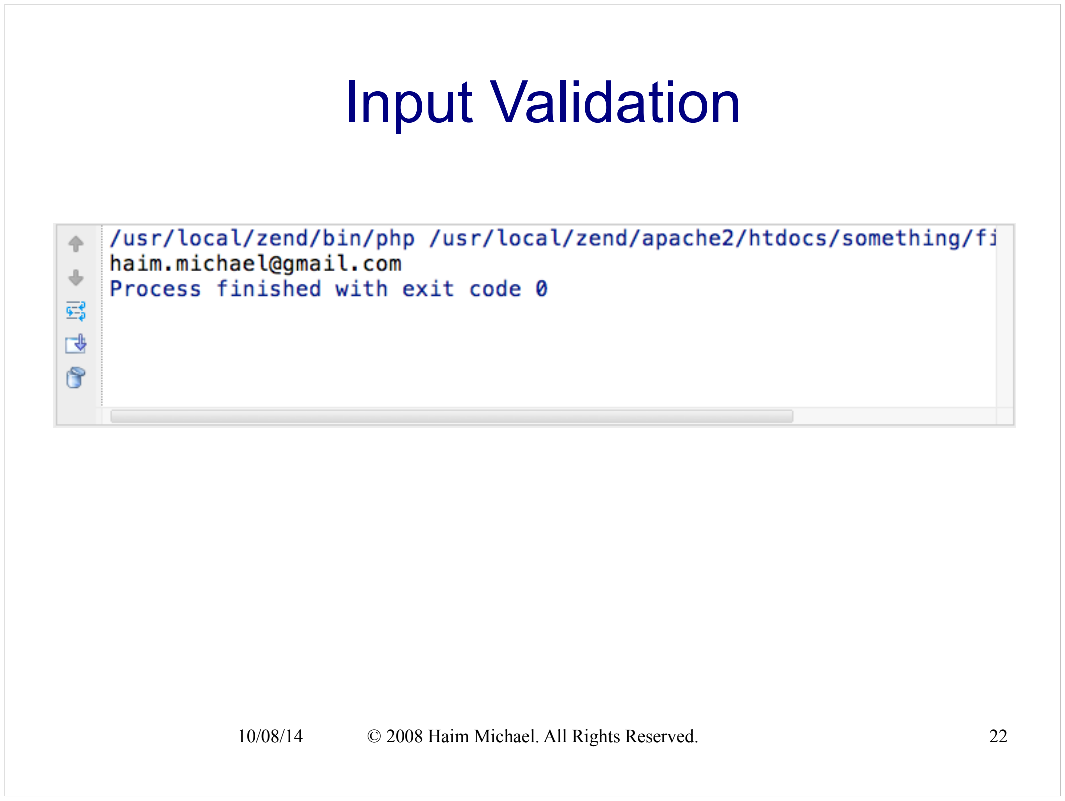
The temporary file is automatically destroyed when the script ends.



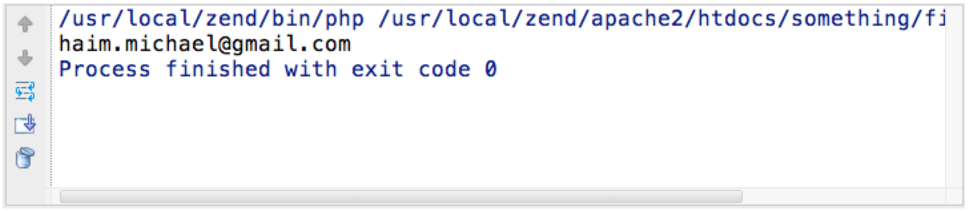
Input Validation

```
<?php
$email_address_dirty = "haim.michael@gmail.com";
if(filter_var($email_address_dirty,FILTER_VALIDATE_EMAIL))
{
    $email_address_clean = $email_address_dirty;
}
echo $email_address_clean;
?>
```

The temporary file is automatically destroyed when the script ends.



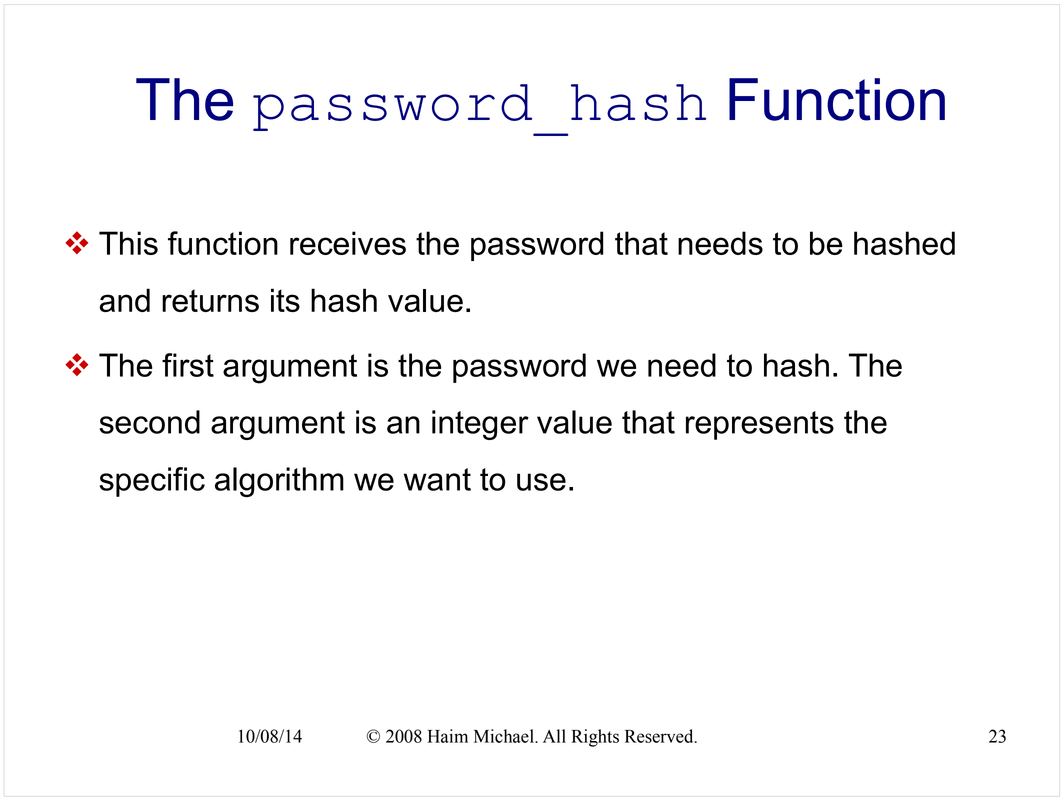
Input Validation



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/fi
haim.michael@gmail.com
Process finished with exit code 0
```

10/08/14 © 2008 Haim Michael. All Rights Reserved. 22

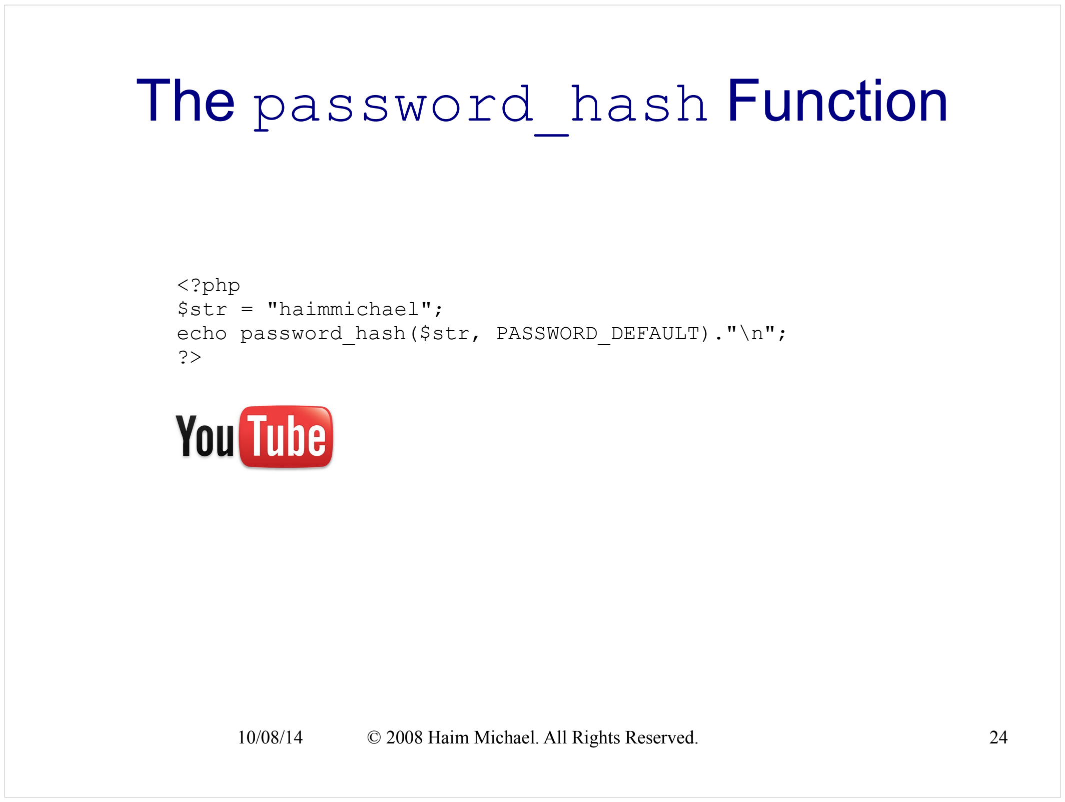
The temporary file is automatically destroyed when the script ends.



The `password_hash` Function

- This function receives the password that needs to be hashed and returns its hash value.
- The first argument is the password we need to hash. The second argument is an integer value that represents the specific algorithm we want to use.

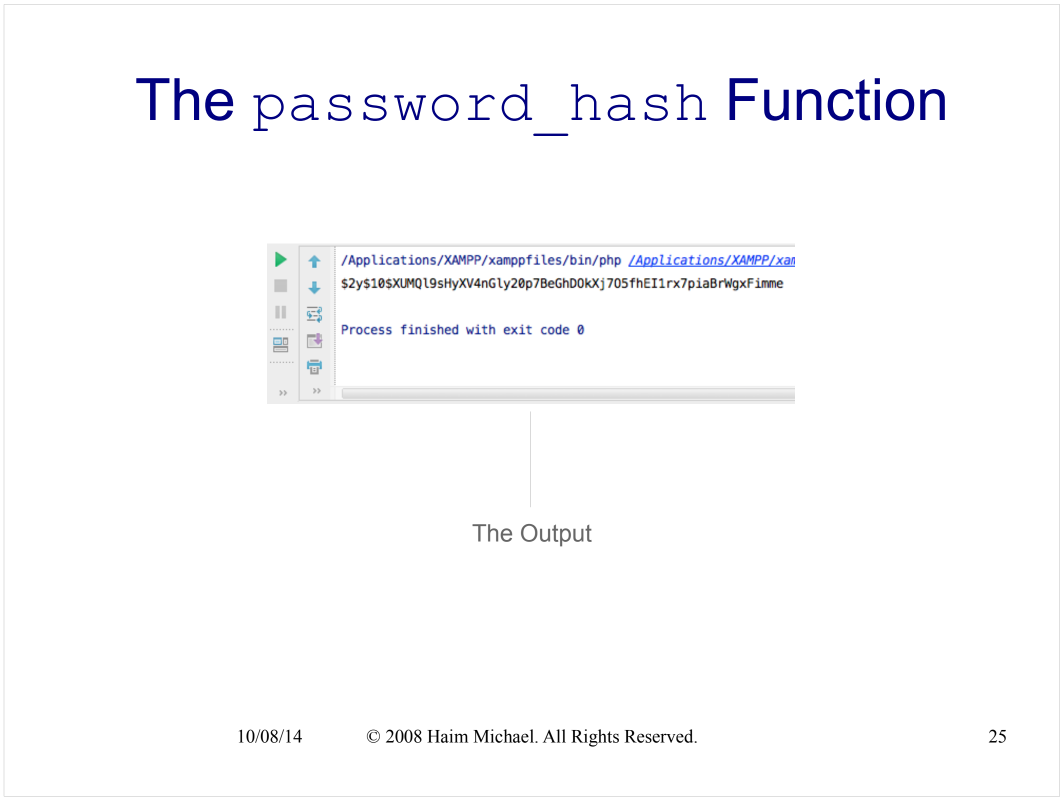
The temporary file is automatically destroyed when the script ends.



The `password_hash` Function

```
<?php
$str = "haimmichael";
echo password_hash($str, PASSWORD_DEFAULT)."\n";
?>
```

The temporary file is automatically destroyed when the script ends.



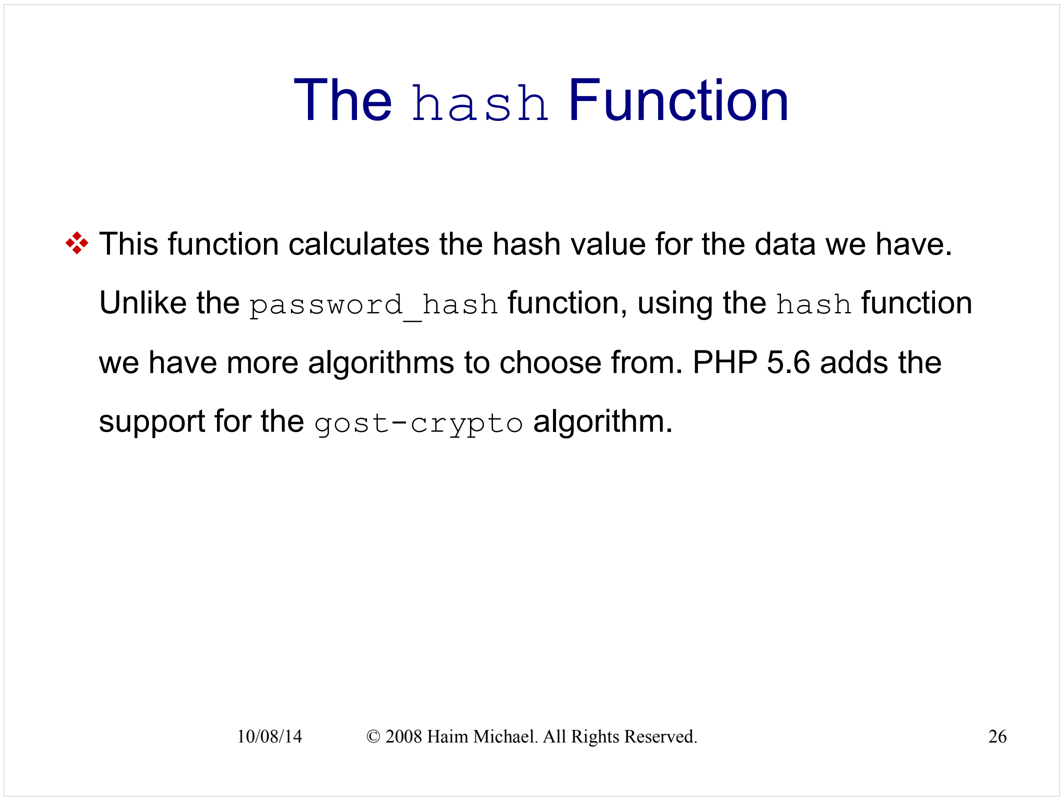
The `password_hash` Function

```
/Applications/XAMPP/xamppfiles/bin/php /Applications/XAMPP/xam
$2y$10$XUMQl9sHyXV4nGly20p7BeGhDOkXj7O5fhEI1rx7piaBrWgxFimme

Process finished with exit code 0
```

The Output

10/08/14 © 2008 Haim Michael. All Rights Reserved. 25

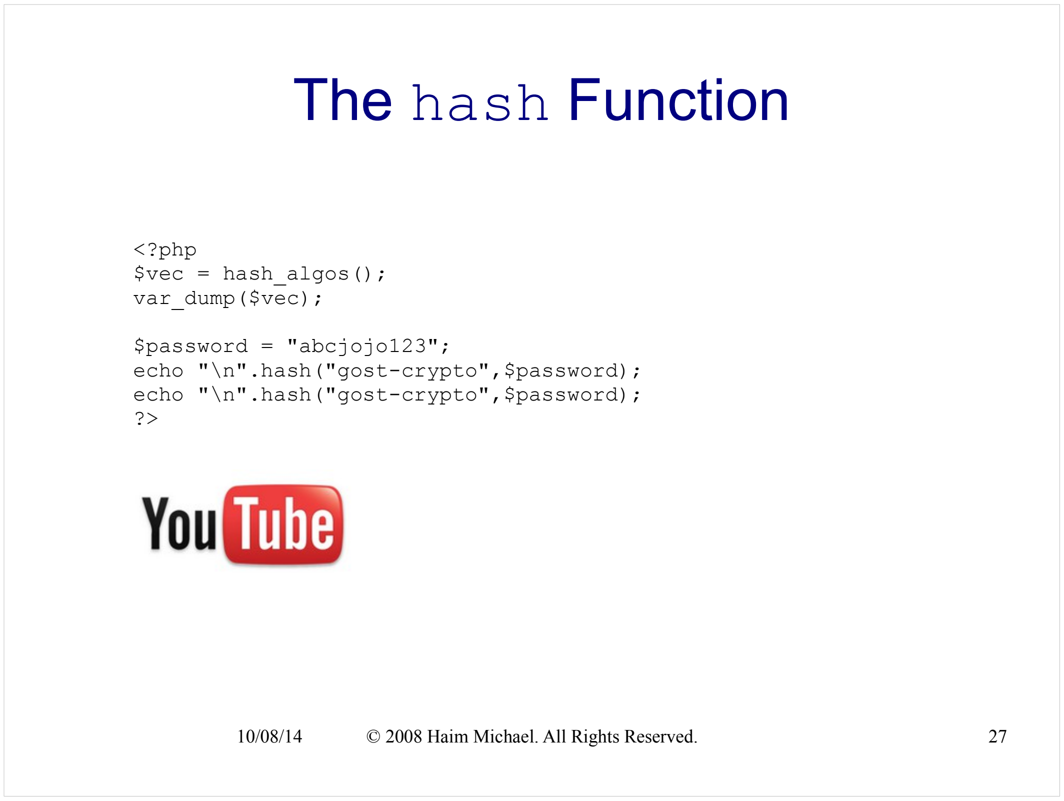
The temporary file is automatically destroyed when the script ends.



The `hash` Function

- This function calculates the hash value for the data we have. Unlike the `password_hash` function, using the `hash` function we have more algorithms to choose from. PHP 5.6 adds the support for the `gost-crypto` algorithm.

The temporary file is automatically destroyed when the script ends.

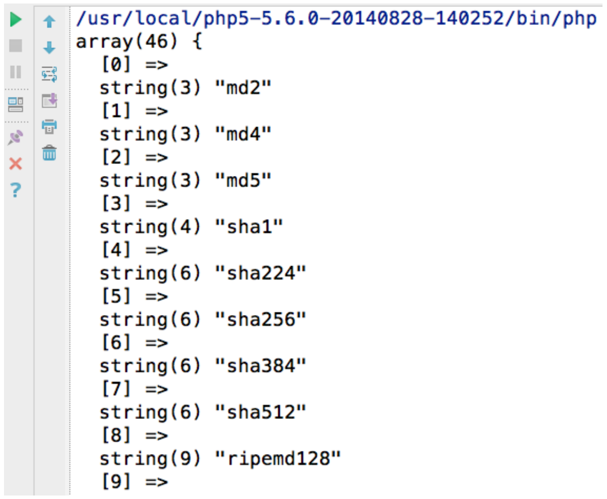


The hash Function

```
<?php
$vec = hash_algos();
var_dump($vec);

$password = "abcjojo123";
echo "\n".hash("gost-crypto",$password);
echo "\n".hash("gost-crypto",$password);
?>
```

The temporary file is automatically destroyed when the script ends.

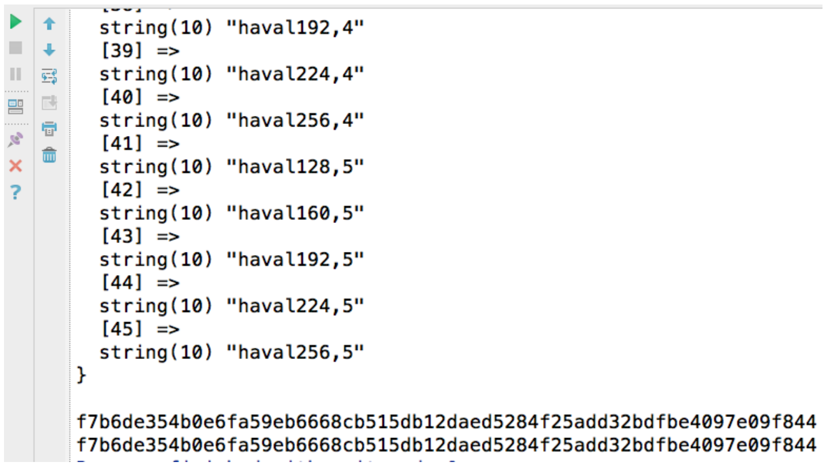
The `hash` Function



```
/usr/local/php5-5.6.0-20140828-140252/bin/php
array(46) {
  [0] =>
  string(3) "md2"
  [1] =>
  string(3) "md4"
  [2] =>
  string(3) "md5"
  [3] =>
  string(4) "sha1"
  [4] =>
  string(6) "sha224"
  [5] =>
  string(6) "sha256"
  [6] =>
  string(6) "sha384"
  [7] =>
  string(6) "sha512"
  [8] =>
  string(9) "ripemd128"
  [9] =>
```

The temporary file is automatically destroyed when the script ends.

The hash Function



```
  string(10) "haval192,4"
  [39] =>
  string(10) "haval224,4"
  [40] =>
  string(10) "haval256,4"
  [41] =>
  string(10) "haval128,5"
  [42] =>
  string(10) "haval160,5"
  [43] =>
  string(10) "haval192,5"
  [44] =>
  string(10) "haval224,5"
  [45] =>
  string(10) "haval256,5"
}

f7b6de354b0e6fa59eb6668cb515db12daed5284f25add32bdfbe4097e09f844
f7b6de354b0e6fa59eb6668cb515db12daed5284f25add32bdfbe4097e09f844
```

The temporary file is automatically destroyed when the script ends.