

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

MCS 260 Lecture 36
Introduction to Computer Science
Jan Verschelde, 19 November 2007

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

GET method is a request for a static resource, such as an HTML page.

Just typing the URL of the requested web page invokes the **GET** method.

POST method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The **GET** and **POST** methods are most commonly used.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

GET method is a request for a static resource, such as an HTML page.

Just typing the URL of the requested web page invokes the GET method.

POST method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The GET and POST methods are most commonly used.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

HTTP (HyperText Transfer Protocol) determines request-response communication between web browser and web server.

Methods of HTTP:

GET method is a request for a static resource, such as an HTML page.

Just typing the URL of the requested web page invokes the **GET** method.

POST method is a request for a dynamic resource, with input parameters of the request contained within the body of the request.

The **GET** and **POST** methods are most commonly used.

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document,
and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1,
other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web
Pages

HTTP: GET and POST
methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document,
and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1,
other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Elements of HTML

HyperText Markup Language

Commonly used elements in HTML documents:

HTML `<HTML>` marks start of HTML document, and `</HTML>` marks the end.

HEAD specifies header information of a document.

TITLE specifies title of the document.

BODY contains the body text of the document.

FONT used to alter font size and color of text.

H1 to display headings of type 1, other heading elements are **H2** and **H3**.

P defines a paragraph.

OL ordered list, unordered list is **UL**.

LI list element in ordered or unordered list.

Learn HTML by looking at source of web pages.

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Accepting Data from Users

To accept data from users,
three elements are generally used:

FORM contains all the code related to a form.
Its purpose is to accept user input in a
systematic and structured manner.

INPUT specifies the code used to create the form
controls that accept user input.

SELECT used to display lists in a form.

Designing and creating interactive web pages
is similar to GUI design.

Accepting Data from Users

To accept data from users, three elements are generally used:

FORM contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

INPUT specifies the code used to create the form controls that accept user input.

SELECT used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.

Accepting Data from Users

To accept data from users, three elements are generally used:

FORM contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

INPUT specifies the code used to create the form controls that accept user input.

SELECT used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.

Accepting Data from Users

To accept data from users, three elements are generally used:

FORM contains all the code related to a form. Its purpose is to accept user input in a systematic and structured manner.

INPUT specifies the code used to create the form controls that accept user input.

SELECT used to display lists in a form.

Designing and creating interactive web pages is similar to GUI design.

the FORM element

definition and syntax

A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

`METHOD` is GET or POST.

`ACTION` is typically used to specify the code that will process the the input data.

Syntax:

```
<FORM METHOD="GET_or_POST" ACTION="file_name" >  
  code_of_the_form  
</FORM>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the FORM element

definition and syntax

A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

METHOD is GET or POST.

ACTION is typically used to specify the code that will process the the input data.

Syntax:

```
<FORM METHOD="GET_or_POST" ACTION="file_name" >  
  code_of_the_form  
</FORM>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the FORM element

definition and syntax

A form is a collection of text boxes, radio buttons, check boxes, and buttons.

Two attributes of a form:

METHOD is GET or POST.

ACTION is typically used to specify the code that will process the the input data.

Syntax:

```
<FORM METHOD="GET_or_POST" ACTION="file_name">  
  code_of_the_form  
</FORM>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the INPUT element

The `INPUT` element is specified inside a `FORM` element.

The `INPUT` elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

`TYPE` specifies type of control to accept user input.

`NAME` specifies name of a control, for identification.

`VALUE` holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

the INPUT element

The INPUT element is specified inside a FORM element.

The INPUT elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

TYPE specifies type of control to accept user input.

NAME specifies name of a control, for identification.

VALUE holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

the INPUT element

The INPUT element is specified inside a FORM element.

The INPUT elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

TYPE specifies type of control to accept user input.

NAME specifies name of a control, for identification.

VALUE holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

the INPUT element

The INPUT element is specified inside a FORM element.

The INPUT elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

TYPE specifies type of control to accept user input.

NAME specifies name of a control, for identification.

VALUE holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

the INPUT element

The INPUT element is specified inside a FORM element.

The INPUT elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

TYPE specifies type of control to accept user input.

NAME specifies name of a control, for identification.

VALUE holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

the INPUT element

The INPUT element is specified inside a FORM element.

The INPUT elements consists of controls, such as text boxes, buttons, radio buttons, and check boxes.

Each of these controls can have attributes:

TYPE specifies type of control to accept user input.

NAME specifies name of a control, for identification.

VALUE holds value entered by user, or default.

Five types of control:

- (1) submit button;
- (2) text boxes;
- (3) radio buttons;
- (4) check boxes;
- (5) combo boxes.

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data

server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

We distinguish between

- ▶ **Client-Side Scripting:** processed by browser
advantage: saves time of the server
- ▶ **Server-Side Scripting:** processed by server
needed for synchronization, modification of data
server time is then *the* time

Python is a powerful server-side scripting language.
The `cgi` module has to be imported,
in order to communicate the data from client.

We distinguish between

- ▶ Client-Side Scripting: processed by browser
advantage: saves time of the server
- ▶ Server-Side Scripting: processed by server
needed for synchronization, modification of data
server time is then *the* time

Python is a powerful server-side scripting language.
The `cgi` module has to be imported,
in order to communicate the data from client.

We distinguish between

- ▶ Client-Side Scripting: processed by browser
advantage: saves time of the server
- ▶ Server-Side Scripting: processed by server
needed for synchronization, modification of data
server time is then *the* time

Python is a powerful server-side scripting language.
The `cgi` module has to be imported,
in order to communicate the data from client.

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

our first CGI script

hello world once more

The script `hello_world.py`:

```
#!/root/Python-2.5.1/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

our first CGI script

hello world once more

The script `hello_world.py`:

```
#!/root/Python-2.5.1/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

our first CGI script

hello world once more

The script `hello_world.py`:

```
#!/root/Python-2.5.1/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

our first CGI script

hello world once more

The script `hello_world.py`:

```
#!/root/Python-2.5.1/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

our first CGI script

hello world once more

The script `hello_world.py`:

```
#!/root/Python-2.5.1/python
print "Content-Type: text/plain\n\n"
print "Hello world!"
```

Explaining the script line by line:

1. line 1 is the path to the Python interpreter
2. line 2 passes MIME information to the browser
MIME = *Multipurpose Internet Mail Extensions*
3. line 3 prints in the browser window

This script should be save in the `cgi-bin` directory, usually in `/var/www/cgi-bin`.

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Local Web Servers

about localhost

MCS 260 L-36

19 November 2007

Dynamic Web
Pages

HTTP: GET and POST
methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

How do we run `hello_world.py` in a browser?

The `localhost` in an URL
refers to the local machine.

The complete path to run `hello_world.py` is
`http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences
and enable "Web Sharing".

Local Web Servers

about localhost

MCS 260 L-36

19 November 2007

Dynamic Web
Pages

HTTP: GET and POST
methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

How do we run `hello_world.py` in a browser?

The `localhost` in an URL
refers to the local machine.

The complete path to run `hello_world.py` is
`http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences
and enable "Web Sharing".

Local Web Servers

about localhost

MCS 260 L-36

19 November 2007

Dynamic Web
Pages

HTTP: GET and POST
methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

How do we run `hello_world.py` in a browser?

The `localhost` in an URL
refers to the local machine.

The complete path to run `hello_world.py` is
`http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences
and enable "Web Sharing".

Local Web Servers

about localhost

MCS 260 L-36

19 November 2007

Dynamic Web
Pages

HTTP: GET and POST
methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

How do we run `hello_world.py` in a browser?

The `localhost` in an URL
refers to the local machine.

The complete path to run `hello_world.py` is
`http://localhost/cgi-bin/hello_world.py`

Leopard (Mac OS X 10.5) comes with Apache:
Go to the sharing panel in system preferences
and enable "Web Sharing".

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

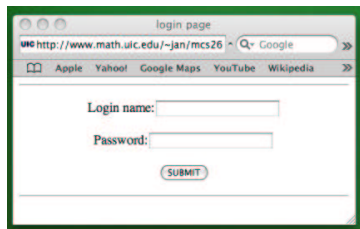
Introduction to CGI

hello localhost

a login page

Python to generate HTML

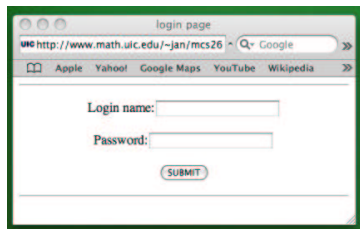
A Login Page



1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action,
the action is implemented with Python script.

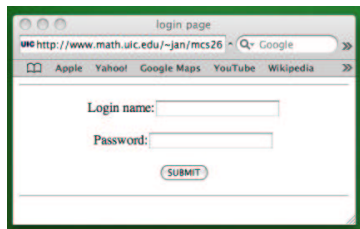
A Login Page



1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action,
the action is implemented with Python script.

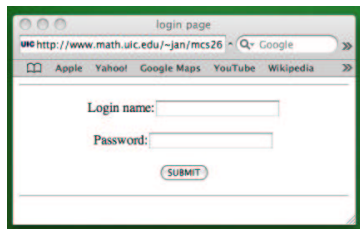
A Login Page



1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action,
the action is implemented with Python script.

A Login Page



1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action,
the action is implemented with Python script.

A Login Page

Dynamic Web Pages

HTTP: GET and POST methods

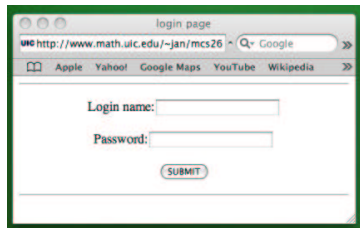
FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML



1. A field to enter login name.
2. A field to enter password.
3. A button to submit login name and password.

The button will trigger an action,
the action is implemented with Python script.

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

the code

the file `loginpage.html`

```
<HTML>
<HEAD>
<TITLE>login page</TITLE>
</HEAD>
<HR><CENTER>
<FORM method="POST"
      action="http://localhost/cgi-bin/validate.py">
<P>Login name:<input type="text"
                    name="login"
                    value = ""></P>
<P>Password:<input type="password"
                   name="password"
                   value = ""></P>
<P><input type = "submit" value="SUBMIT"></P>
</FORM>
<HR>
</CENTER>
</BODY>
</HTML>
```

Dynamic Web Pages

HTTP: GET and POST methods

FORM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```

How do we recover the user data?

When the user enters data, an instance of `cgi.FieldStorage()` is created.

This object is similar to a dictionary:

1. the keys are the names of the form items;
2. the values are what the user has entered.

In `validate.py`, we could print input data:

```
fs = cgi.FieldStorage()
print fs['login'].value
print fs['password'].value
```

Dynamic Web Pages

HTTP: GET and POST methods
FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost
a login page
Python to generate HTML

Dynamic Web Pages

HTTP: GET and POST methods

FROM to accept user data
server-side scripting

Introduction to CGI

hello localhost

a login page

Python to generate HTML

Displaying Current Date and Time

Running the script `showtime.py` the prompt:

```
<HTML>
<HEAD><TITLE>current date and time</TITLE></HEAD>
<BODY>
Mon Nov 19 12:04:16 2007
</BODY></HTML>
```

The Python script `showtime.py` produces HTML code, containing computed results, e.g. the current time:

```
>>> import time
>>> time.ctime()
'Mon Nov 19 12:04:16 2007'
```

Python to generate HTML

```
#!/root/Python-2.5.1/python
import cgi
import time

def printHeader(title):
    print ""

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>"" % title

printHeader("current date and time")
print time.ctime(time.time())
print "</BODY></HTML>"
```

Python to generate HTML

```
#!/root/Python-2.5.1/python
import cgi
import time

def printHeader(title):
    print """

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>""" % title

printHeader("current date and time")
print time.ctime(time.time())
print "</BODY></HTML>"
```

Python to generate HTML

```
#!/root/Python-2.5.1/python
import cgi
import time

def printHeader(title):
    print """

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>""" % title

printHeader("current date and time")
print time.ctime(time.time())
print "</BODY></HTML>"
```

Python to generate HTML

```
#!/root/Python-2.5.1/python
import cgi
import time

def printHeader(title):
    print ""

<HTML>
<HEAD><TITLE>%s</TITLE></HEAD>
<BODY>"" % title

printHeader("current date and time")
print time.ctime(time.time())
print "</BODY></HTML>"
```

We covered more of chapter 10 in *Making Use of Python*.

Assignments:

1. Make a web interface for the temperature converter (see the first project).
2. Write a CGI script in Python for a currency converter.

There will be a quiz in lab session this Tuesday!