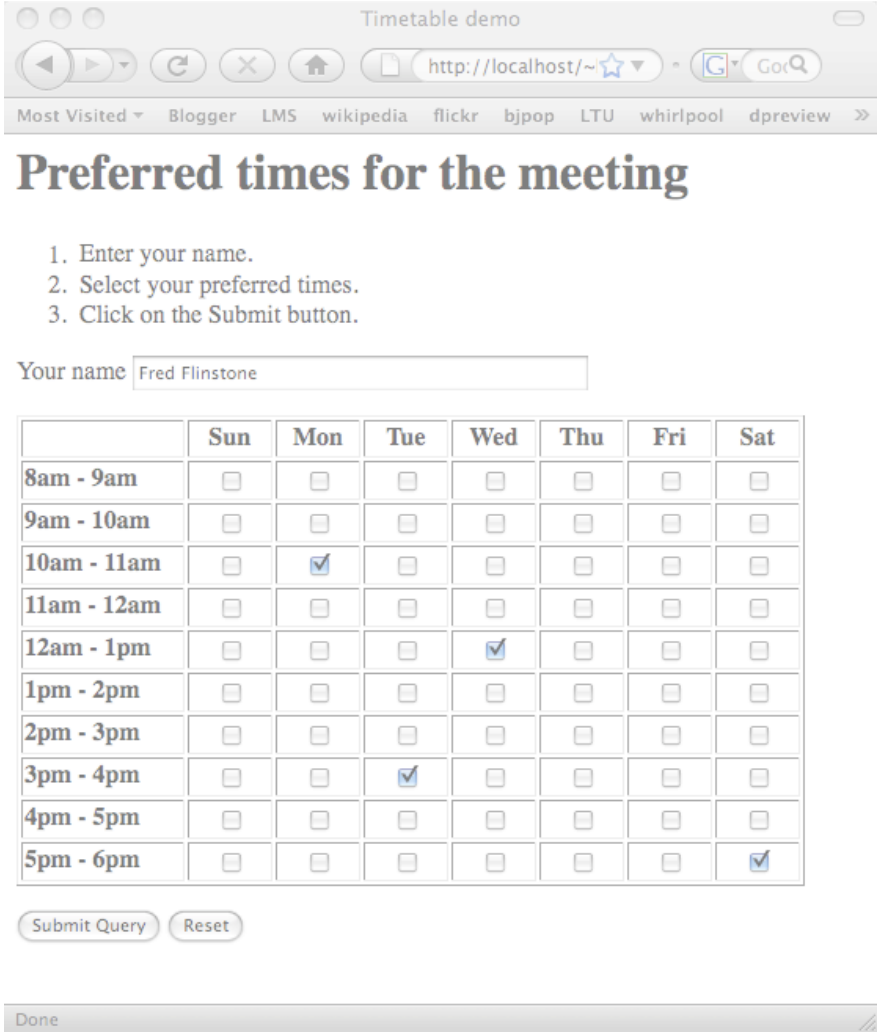
HTML forms

A user interface to CGI applications



Outline

- A simple example form.
- GET versus POST.
- cgi.escape().
- Input controls.

A very simple form

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

          <form method="post" action="http://localhost/~bjpop/env.py">
              <input type="text" name="foo" />
              <input type="submit" />
          </form>

   </body>
</html>
```

The simple form looks like this in Safari

A very simple form

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>
```

```
<form method="post" action="http://localhost/~bjpop/env.py">
    <input type="text" name="foo" />
    <input type="submit" />
</form>
```

```
   </body>
</html>
```

Usual HMTL stuff. Note the use of “xhtml” syntax. Probably should declare DOCTYPE and the xhtml namespace for validation purposes, but we are keeping it short for the sake of these slides.

A very simple form

The form element, containing two control elements.

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/env.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

A very simple form

The first control element is a text input box.
The name of the control is *foo*.

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/env.py">
            <input type="text" name="foo" />
            <input type="submit" />
         </form>

   </body>
</html>
```

A very simple form

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/env.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

The second control element is a submit button. When the user clicks on the button, the form data is sent to the server.

A very simple form

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/env.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

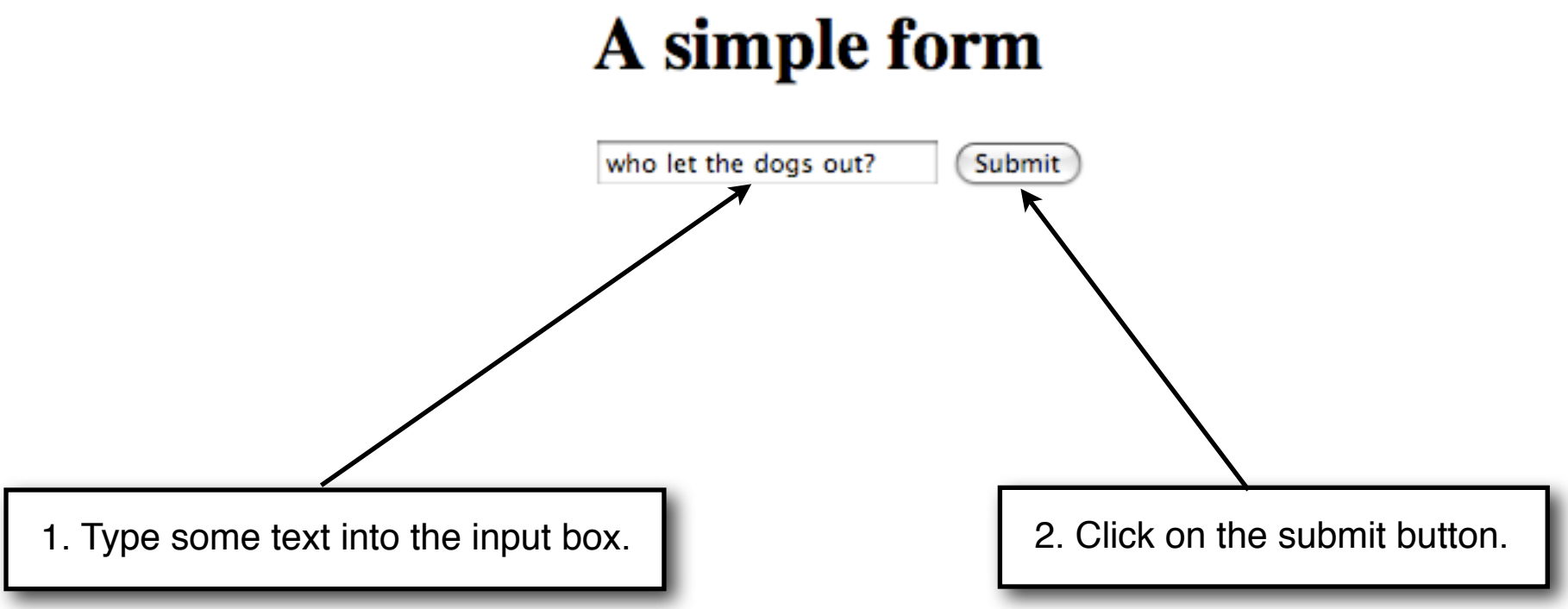
The method attribute of the form element says how the form data is sent to the server. It can either be “get” or “post”. It determines the underlying HTTP request method used by the client.

A very simple form

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/env.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

The action attribute of the form element determines which CGI script the form data should be sent to for processing.

The form in action



Output from the env.py CGI script.

```
HTTP_REFERER = http://localhost/~bjpop/forms-lecture/simpleform.html
SERVER_SOFTWARE = Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
SCRIPT_NAME = /~bjpop/env.py
SERVER_SIGNATURE =
REQUEST_METHOD = POST
SERVER_PROTOCOL = HTTP/1.1
QUERY_STRING =
PATH = /usr/bin:/bin:/usr/sbin:/sbin
CONTENT_LENGTH = 27
HTTP_USER_AGENT = Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_4; en-us)  ...
HTTP_CONNECTION = keep-alive
SERVER_NAME = localhost
REMOTE_ADDR = ::1
SERVER_PORT = 80
SERVER_ADDR = ::1
DOCUMENT_ROOT = /Library/WebServer/Documents
SCRIPT_FILENAME = /Users/bjpop/Sites/env.py
SERVER_ADMIN = you@example.com
HTTP_HOST = localhost
REQUEST_URI = /~bjpop/env.py
HTTP_ACCEPT = text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
GATEWAY_INTERFACE = CGI/1.1
REMOTE_PORT = 51901
HTTP_ACCEPT_LANGUAGE = en-us
CONTENT_TYPE = application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING = gzip, deflate
stdin = foo=who+let+the+dogs+out%3F
```

```
HTTP_REFERER = http://localhost/~bjpop/forms-lecture/simpleform.html
SERVER_SOFTWARE = Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
SCRIPT_NAME = /~bjpop/env.py
SERVER_SIGNATURE =
REQUEST_METHOD = POST
SERVER_PROTOCOL = HTTP/1.1
QUERY_STRING =
PATH = /usr/bin:/bin:/usr/sbin:/sbin
CONTENT_LENGTH = 27
HTTP_USER_AGENT = Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_4; en-us)  ...
HTTP_CONNECTION = keep-alive
SERVER_NAME = localhost
REMOTE_ADDR = ::1
SERVER_PORT = 80
SERVER_ADDR = ::1
DOCUMENT_ROOT = /Library/WebServer/Documents
SCRIPT_FILENAME = /Users/bjpop/Sites/env.py
SERVER_ADMIN = you@example.com
HTTP_HOST = localhost
REQUEST_URI = /~bjpop/env.py
HTTP_ACCEPT = text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
GATEWAY_INTERFACE = CGI/1.1
REMOTE_PORT = 51901
HTTP_ACCEPT_LANGUAGE = en-us
CONTENT_TYPE = application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING = gzip, deflate
stdin = foo=who+let+the+dogs+out%3F
```

The form used `method="post"`, so the REQUEST_METHOD is POST.

The QUERY_STRING is empty because POST sends the data via standard input (stdin).

Data from the input box called “foo”. Note that the data is encoded. Spaces have been replaced with +, and the question mark is encoded as %3F.

GET versus POST

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="get" action="http://localhost/~bjpop/env.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

Change the method to “get”.

```
HTTP_REFERER = http://localhost/~bjpop/forms-lecture/simpleform.html
SERVER_SOFTWARE = Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.7l DAV/2
SCRIPT_NAME = /~bjpop/env.py
SERVER_SIGNATURE =
REQUEST_METHOD = GET
SERVER_PROTOCOL = HTTP/1.1
QUERY_STRING = foo=who+let+the+dogs+out%3F
PATH = /usr/bin:/bin:/usr/sbin:/sbin
HTTP_USER_AGENT = Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_4; en-us) AppleWebKit/525.18
(KHTML, like Gecko) Version/3.1.2 Safari/525.20.1
HTTP_CONNECTION = keep-alive
SERVER_NAME = localhost
REMOTE_ADDR = ::1
SERVER_PORT = 80
SERVER_ADDR = ::1
DOCUMENT_ROOT = /Library/WebServer/Documents
SCRIPT_FILENAME = /Users/bjpop/Sites/env.py
SERVER_ADMIN = you@example.com
HTTP_HOST = localhost
HTTP_CACHE_CONTROL = max-age=0
REQUEST_URI = /~bjpop/env.py?foo=who+let+the+dogs+out%3F
HTTP_ACCEPT = text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
GATEWAY_INTERFACE = CGI/1.1
REMOTE_PORT = 52008
HTTP_ACCEPT_LANGUAGE = en-us
HTTP_ACCEPT_ENCODING = gzip, deflate
stdin =
```

Output from the env.py CGI script, when the form uses the “get” method instead of the “post” method.

GET versus POST

- At the time of writing POST is not working properly on IVLE (oops).
- Hopefully it will be fixed soon.
- In the meantime, just use the “get” method for forms.
- In general, “get” is recommended only for *idempotent* requests. A request is idempotent if it returns the same result every time it is run.
- An idempotent request does not have “side effects”. That is, it does not change the state of the server. For example, it does not cause a file on the server to be modified.
- Why do you think this is this recommended?

An alternative CGI script for the action

Assume that this script is called "foo.py".

```
import cgi

print 'Content-Type: text/html'
print

store = cgi.FieldStorage()

print '<html><head></head><body><h1>'

if 'foo' in store:
    print store.getvalue('foo')

print '</h1></body></html>'
```

Modify the action attribute to point to “foo.py”

```
<html>
   <head>
      <title>A simple form</title>
   </head>
   <body>
      <h1>A simple form</h1>

         <form method="post" action="http://localhost/~bjpop/foo.py">
             <input type="text" name="foo" />
             <input type="submit" />
         </form>

   </body>
</html>
```

Let’s send the form data to “foo.py” instead of “env.py”.

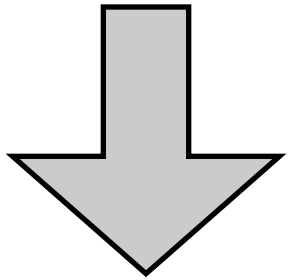
The foo.py script in action.

who let the dogs out?

The foo.py script in action.

This is really the HTML:

```
<html><head></head><body><h1>
who let the dogs out?
</h1></body></html>
```



who let the dogs out?

The foo.py script in action.

Notice that the string is decoded automatically by the Python cgi library.

who let the dogs out?

Warning!

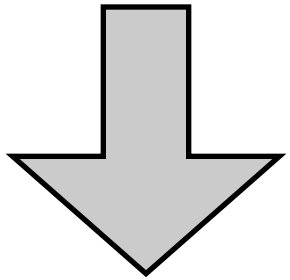
```
import cgi

print 'Content-Type: text/html'
print

store = cgi.FieldStorage()

print '<html><head></head><body><h1>'

if 'foo' in store:
    print store.getvalue('foo')

print '</h1></body></html>'
```

Danger! Danger! What happens if the user types html tags in the input box?

The user types HTML tags into the input text.

This is really the HTML:

```
<html><head></head><body><h1>
<i>who</i> let the dogs out?
</h1></body></html>
```



***who* let the dogs out?**

cgi.escape()

```
import cgi

print 'Content-Type: text/html'
print

store = cgi.FieldStorage()

print '<html><head></head><body><h1>'

if 'foo' in store:
    print cgi.escape(store.getvalue('foo'))

print '</h1></body></html>'
```

Now it is safe.

cgi.escape()

This is really the HTML:

```
<html><head></head><body><h1>
&lt;i&gt;who&lt;/i&gt; let the dogs out?
</h1></body></html>
```

who let the dogs out?

Input controls

- HTML forms provide several input controls, for example:
  - text input boxes (single line and multi-line)
  - checkboxes
  - radio buttons
  - submit button
  - reset button
  - buttons (other than submit and reset)
  - menus
  - hidden fields

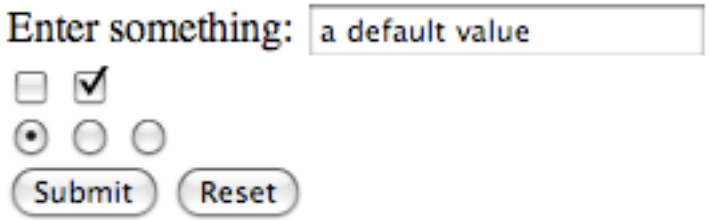
Input controls

- We don’t have time to cover all the controls in the lecture.
- You will try them out in next week’s workshop.
- See: `http://www.w3.org/TR/html401/interact/forms.html`

A form with a few different controls

```
<form method="get" action="http://localhost/~bjpop/env.py">
    Enter something:
    <input type="text" name="mytextbox" value="a default value" />
    <br />
    <input type="checkbox" name="mycheckbox1" />
    <input type="checkbox" name="mycheckbox2" checked="checked" />
    <br />
    <input type="radio" name="myradio" value="value1" checked="checked" />
    <input type="radio" name="myradio" value="value2" />
    <input type="radio" name="myradio" value="value3" />
    <br />
    <input type="submit" />
    <input type="reset" />
</form>
```

The form looks like this in Safari



A form with a few different controls

```
<form method="get" action="http://localhost/~bjpop/env.py">
    Enter something:
    <input type="text" name="mytextbox" value="a default value" />
    <br />
    <input type="checkbox" name="mycheckbox1" />
    <input type="checkbox" name="mycheckbox2" checked="checked" />
    <br />
    <input type="radio" name="myradio" value="value1" checked="checked" />
    <input type="radio" name="myradio" value="value2" />
    <input type="radio" name="myradio" value="value3" />
    <br />
    <input type="submit" />
    <input type="reset" />
</form>
```

Controls are identified by their name.

Radio buttons (and also check boxes) are grouped together by using the same name.

A form with a few different controls

```
<form method="get" action="http://localhost/~bjpop/env.py">
    Enter something:
    <input type="text" name="mytextbox" value="a default value" />
    <br />
    <input type="checkbox" name="mycheckbox1" />
    <input type="checkbox" name="mycheckbox2" checked="checked" />
    <br />
    <input type="radio" name="myradio" value="value1" checked="checked" />
    <input type="radio" name="myradio" value="value2" />
    <input type="radio" name="myradio" value="value3" />
    <br />
    <input type="submit" />
    <input type="reset" />
</form>
```

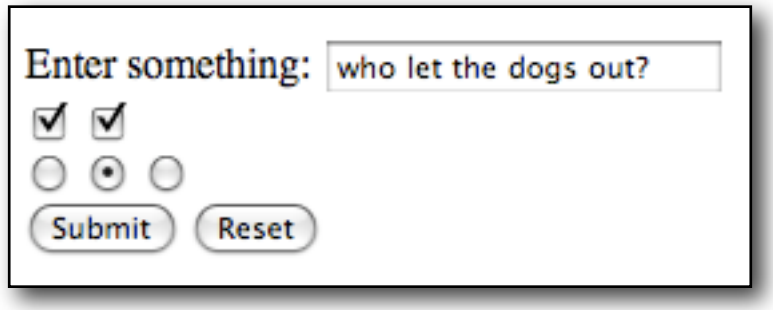
Some controls, such as text boxes, can have default values.

A form with a few different controls

```
<form method="get" action="http://localhost/~bjpop/env.py">
    Enter something:
    <input type="text" name="mytextbox" value="a default value" />
    <br />
    <input type="checkbox" name="mycheckbox1" />
    <input type="checkbox" name="mycheckbox2" checked="checked" />
    <br />
    <input type="radio" name="myradio" value="value1" checked="checked" />
    <input type="radio" name="myradio" value="value2" />
    <input type="radio" name="myradio" value="value3" />
    <br />
    <input type="submit" />
    <input type="reset" />
</form>
```

Checkboxes and radio buttons can be checked by default.

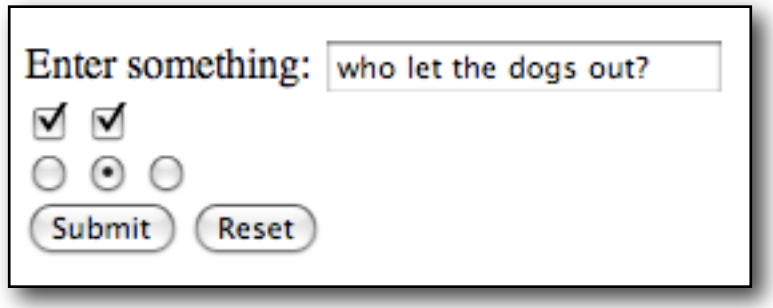
Let's submit this form to env.py



sent to env.py using the "get" method

```
QUERY_STRING =
mytextbox=who+let+the+dogs+out%3F&mycheckbox1=on&mycheckbox2=on&myradio=value2
```

Let's submit this form to env.py

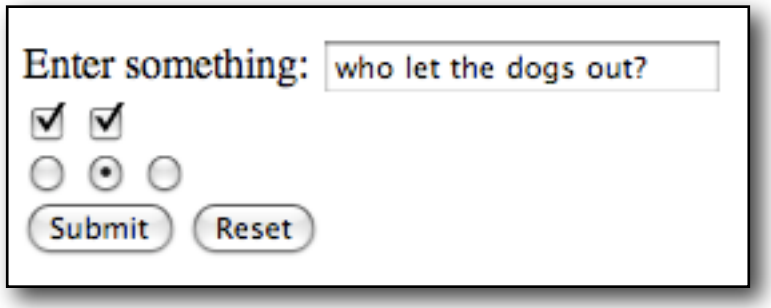


sent to env.py using the "get" method

```
QUERY_STRING =
mytextbox=who+let+the+dogs+out%3F&mycheckbox1=on&mycheckbox2=on&myradio=value2
```

See how the names of the controls are sent in the QUERY_STRING

Let's submit this form to env.py



sent to env.py using the "get" method

```
QUERY_STRING =
mytextbox=who+let+the+dogs+out%3F&mycheckbox1=on&mycheckbox2=on&myradio=value2
```

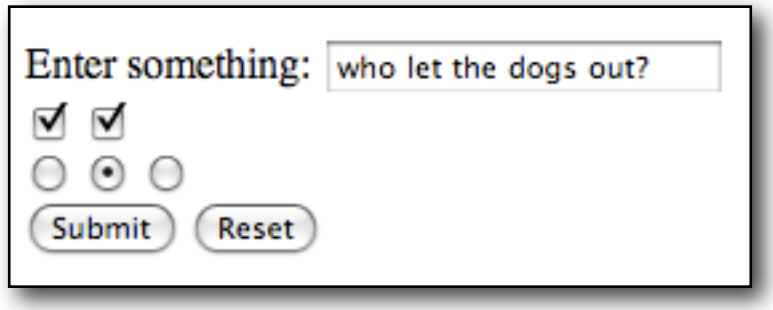
Checkboxes which are selected get the value "on" by default, but you can also use custom values via the `value` attribute.

A (slightly) more interesting CGI script for our form

Assume that this script is called "bar.py".

```
import cgi

print 'Content-Type: text/html'
print

store = cgi.FieldStorage()

print '<html><head></head><body><table border="1">'
print '<tr><th>name</th><th>value</th></tr>'

for var in store:
    value = cgi.escape(store.getvalue(var))
    print '<tr><td>%s</td><td>%s</td></tr>' % (var, value)

print '</table></body></html>'
```

Let's submit this form to bar.py



sent to bar.py using the "get" method

name	value
mytextbox	who let the dogs out?
mycheckbox1	on
mycheckbox2	on
myradio	value2

Remarks

- A single HTML document can contain multiple forms, and hence multiple submit buttons.
- However, forms cannot be nested. That is, one form cannot be inside another.
- It is possible to write the CGI script to produce a form as output, and also act as the recipient of the form data (by being the “action” of the form).

Homework - make a form that looks like this

Preferred times for the meeting

1. Enter your name.
2. Select your preferred times.
3. Click on the Submit button.

Your name

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
8am - 9am	☐	☐	☐	☐	☐	☐	☐
9am - 10am	☐	☐	☐	☐	☐	☐	☐
10am - 11am	☐	☐	☐	☐	☐	☐	☐
11am - 12am	☐	☐	☐	☐	☐	☐	☐
12am - 1pm	☐	☐	☐	☐	☐	☐	☐
1pm - 2pm	☐	☐	☐	☐	☐	☐	☐
2pm - 3pm	☐	☐	☐	☐	☐	☐	☐
3pm - 4pm	☐	☐	☐	☐	☐	☐	☐
4pm - 5pm	☐	☐	☐	☐	☐	☐	☐
5pm - 6pm	☐	☐	☐	☐	☐	☐	☐

Submit Reset

In the next lecture

- Cookies for tracking user identity.