CMPT 165
INTRODUCTION TO THE INTERNET AND THE WORLD WIDE WEB




Unit 8
HTML Forms and Basic CGI

SFU

Learning Objectives

In this unit you will learn the following.

- **Understand** the actions performed by the client and the server when a dynamic request is made.
- **Create** HTML forms to capture user input.
- **Create** Python programs to generate web pages, using a user's input.
- **Convert** the types of values in Python as appropriate for calculations and output.

Topics

1. Dynamic Web Pages, CGI
2. HTML Forms
3. HTML Input Tags
4. Python CGI Library
5. Add Script In-Class Exercise
6. Divide Script In-Class Exercise

Fetching, Step 1

URL: `http://cmpt165.csil.sfu.ca/~smakonin/iphone.html`

The web browser contacts the server specified in the Server (host + domain), `cmpt165.csil.sfu.ca`.

It asks for the file with path:

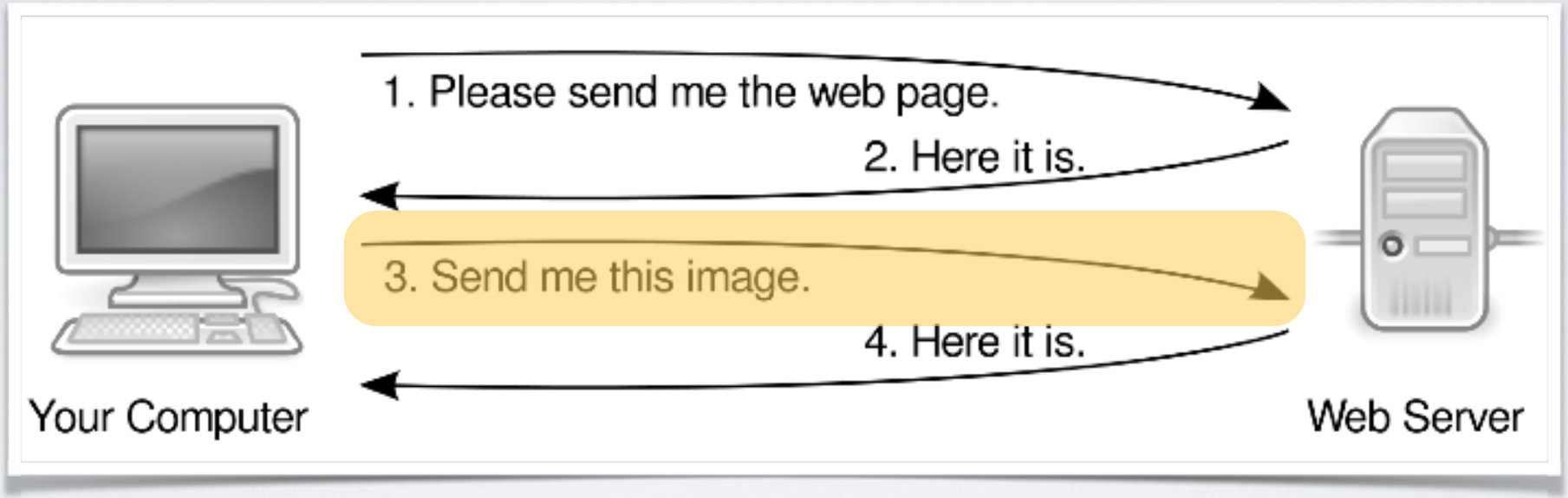
`/home/smakonin/public_html/iphone.html`

Source: CMPT 165 Course Study Guide, p. 23

Fetching, Step 2

The server responds with an **OK** message, indicating that the page has been found and will be sent. It indicates that the MIME type of the file is **text/html**—it's an HTML page. Then it sends the contents of the file, so the browser can display it.

Fetching, Step 3



The browser notices that the web page contains an image with URL:

`http://cmpt165.csil.sfu.ca/~smakonin/iphone.png`

It asks for the file with path:

`/home/smakonin/public_html/iphone.png`

Fetching, Step 4

The server again responds with an **OK** and gives the MIME type **image/png**, which indicates a JPEG format image. Then it sends the actual contents of the image file.

Finally the full webpage is displayed!

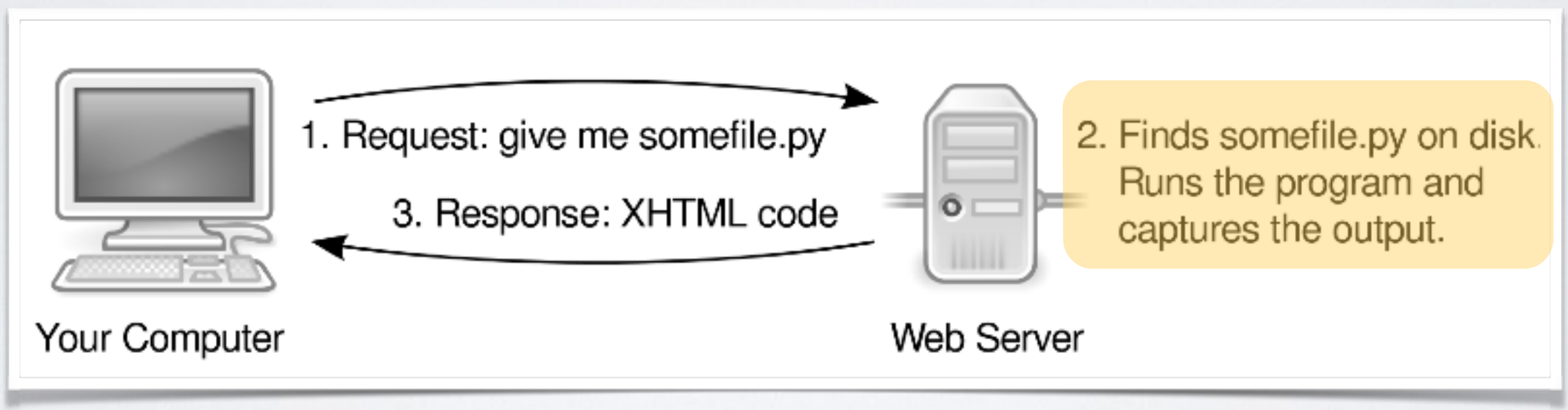
Requesting Static HTML

- Static means the HTML never changes
  - unless you upload a new version
- Web server simple reads file's contents then sends
- Files end with the extension .html (usually)

WHAT IF WE HAVE LOTS OF PAGES THAT HAVE THE SAME STRUCTURE, BUT ONLY CONTACT CHANGES?

Source: CMPT 165 Course Study Guide, p. 121

Requesting Dynamic HTML



Source: CMPT 165 Course Study Guide, p. 122

CGI, Web Scripts

def. Common Gateway Interface, an interface programming languages can use to produce dynamic HTML.

- On the web server we have `here.py`:

```
print "Content-type: text/html"
print
print "<html><head>"
print "<title>Python did this</title>"
print "</head><body>"
print "<p>Here I am</p>"
print "</body></html>"
```

The MIME Type

Blank line mean no more header information HTML data to follow.

`print ≡ print ""`

- Nice and easy with python, like printing text to the screen.

Source: CMPT 165 Course Study Guide, p. 122

CGI, Web Scripts

- Server is configured to know that file with extension:
  - `.html` send to clients
  - `.py` run and send output to client
  - `.cgi` run and send output to client
    - e.g. compiled C/C++
    - Very common in the "good old days".
- browser doesn't care where the HTML came from
  - it just displays it

Content Type

```
print "Content-type: text/html"
```

- Again, a way for the browser to determine the content being returned from a request.
- **BUT,** the `content type` can be any valid MIME.
- If you have a sophisticated program you can generate an image (e.g. chart, graph) *on the fly* and that returns binary image code. The MIME time might look like:

```
print "Content-type: image/png"
```

Generating Content

- Any code that runs using command line Python
- Will also run when used as CGI Python
- So the following statements would work (`plain.py`):

```
print "Content-type: text/plain"
print
age = 2
print age * 5
print "The answer is: ", age
print age ** age
```

HTML Forms

`<form> … </form>`

- Forms are a block element that contain other elements.
- There are special elements to capture user input.
- When the form is filled the user will click on a <u>button</u> that will **submit** the form data to a CGI script.
- Button names are commonly called:
    - **submit**, **login**, **save**, **update**

Form Attributes

`action:` action to be performed when the form is submitted, usually the relative URL of the CGI script.

```
<form action="action.py">
```

`method:` specifies the HTTP method (**`GET`** or **`POST`**) to be used when submitting the forms

```
<form action="action.py" method="POST">
```

`name:` each input field must have a name attribute.

```
<input type="text" name="lastname" value="Mouse">
```

Get or Set?

You can use **GET** (the default method):

- Passive submission (like a search engine query).
- **DO NOT** use with sensitive information.
- Form data will be visible in the page address.
- Browsers set size limitations for URL size.

```
http://www.myserver.com/forms/action.py?firstname=Mickey&lastname=Mouse
```

You should use **POST**:

- **Best** for sensitive information (password).
- This means data can be with encrypted (https://).
- Submitted data is not visible in the page address.

Grouping Form Data

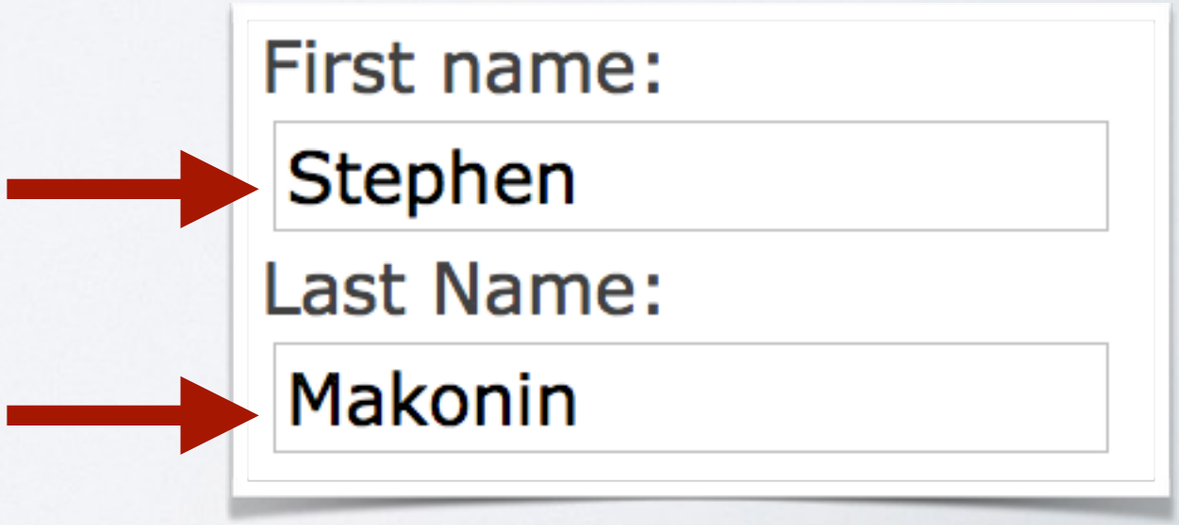
A way to group common fields in long compacted forms.
The design principles of contrast and proximity?

```
<form action="action_page.py">
 <fieldset>
    <legend>Personal information:</legend>
    …
 </fieldset>
</form>
```

Text Input

An input field for one line/word/phrase of text.

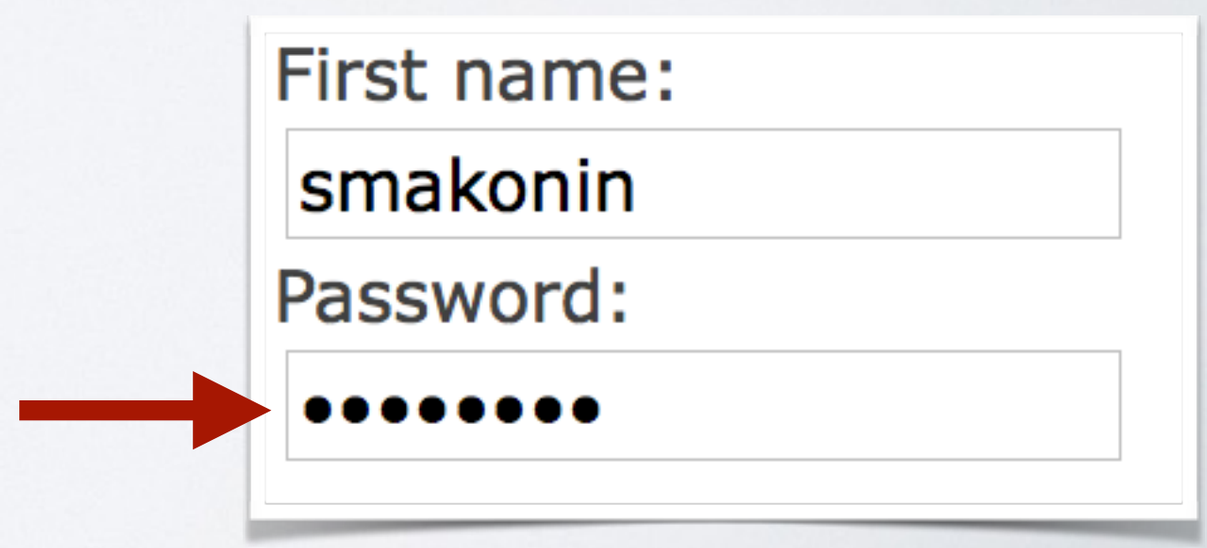
```
<form action="action_page.py">
 First name:<br>
 <input type="text" name="firstname" />
 <br>
 Last name:<br>
 <input type="text" name="lastname" />
</form>
```



Password Input

Similar to text input but the characters are masked (shown as asterisks or circles).

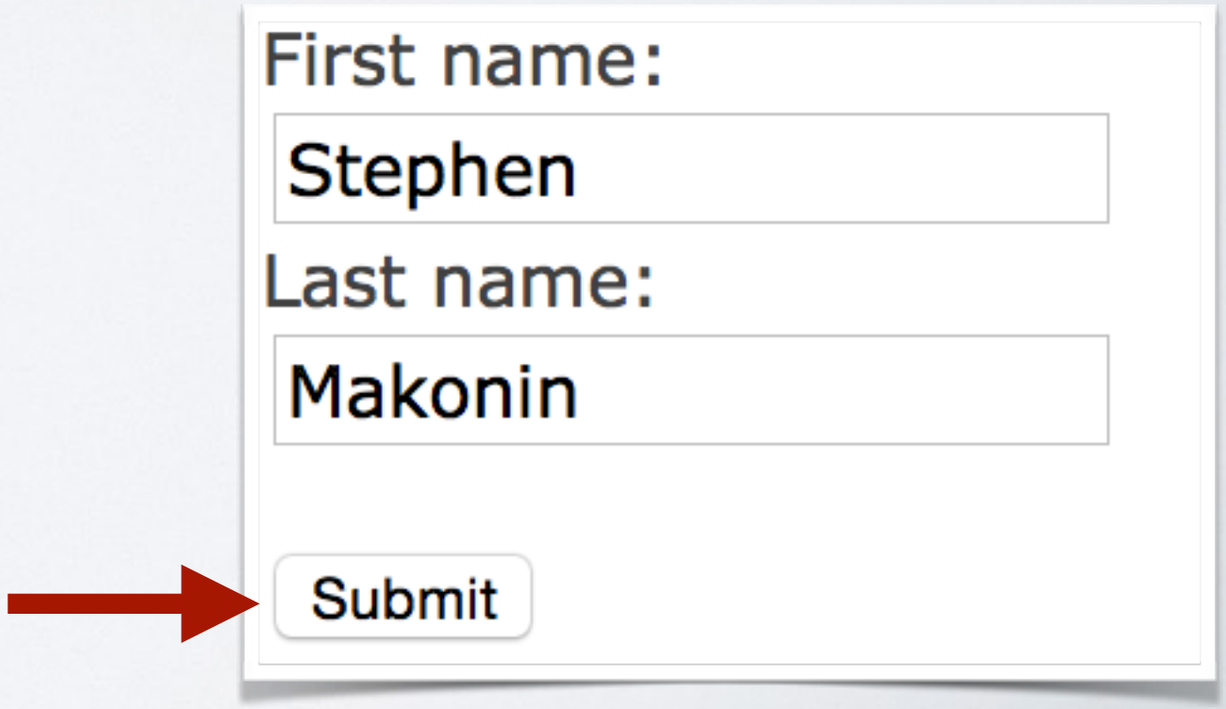
```
<form action="action_page.py">
 User name:<br>
  <input type="text" name="username" />
  <br>
  User password:<br>
  <input type="password" name="psw" />
</form>
```



Submit Button

Submits the form to the .

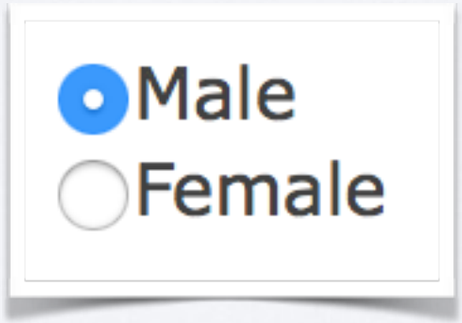
```
<form action="action_page.py">
 …
 Last name:<br>
 <input type="text" name="lastname" value="Mouse" />
 <br><br>
 <input type="submit" value="Submit" />
</form>
```



Radio Buttons

Used to limit the selection of options to one option.

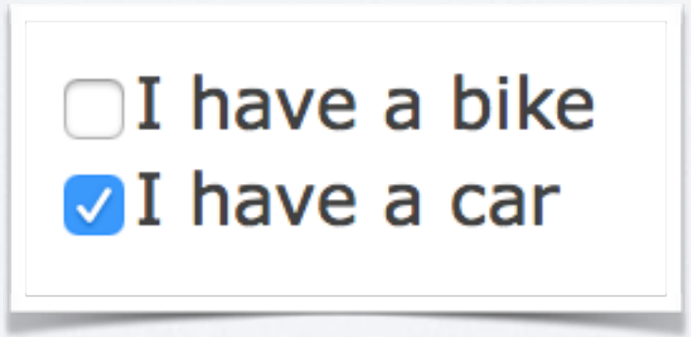
```
<form action="action_page.py">
 <input type="radio" name="gender" value="m" checked />
 Male
 <br>
 <input type="radio" name="gender" value="f" />
 Female
</form>
```



Checkbox

Used to check off zero or more options.

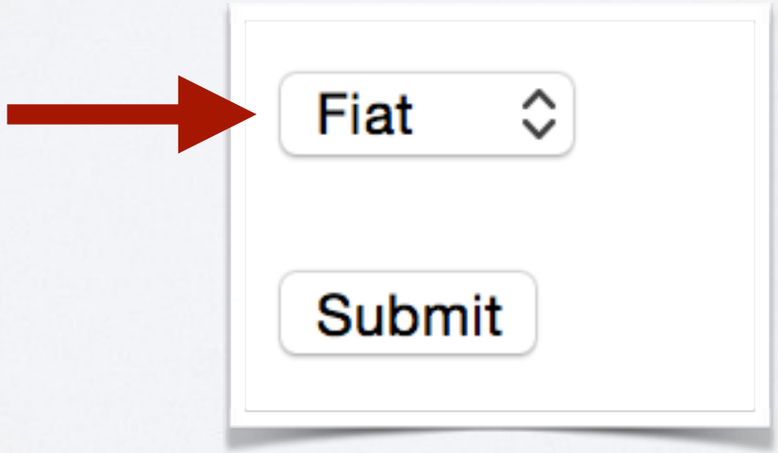
```
<form action="action_page.py">
 <input type="checkbox" name="vehicle" value="Bike" />
 I have a bike
 <br>
 <input type="checkbox" name="vehicle" value="Car" />
 I have a car
</form>
```



Select - Option

Select one option from a drop-down list of options.

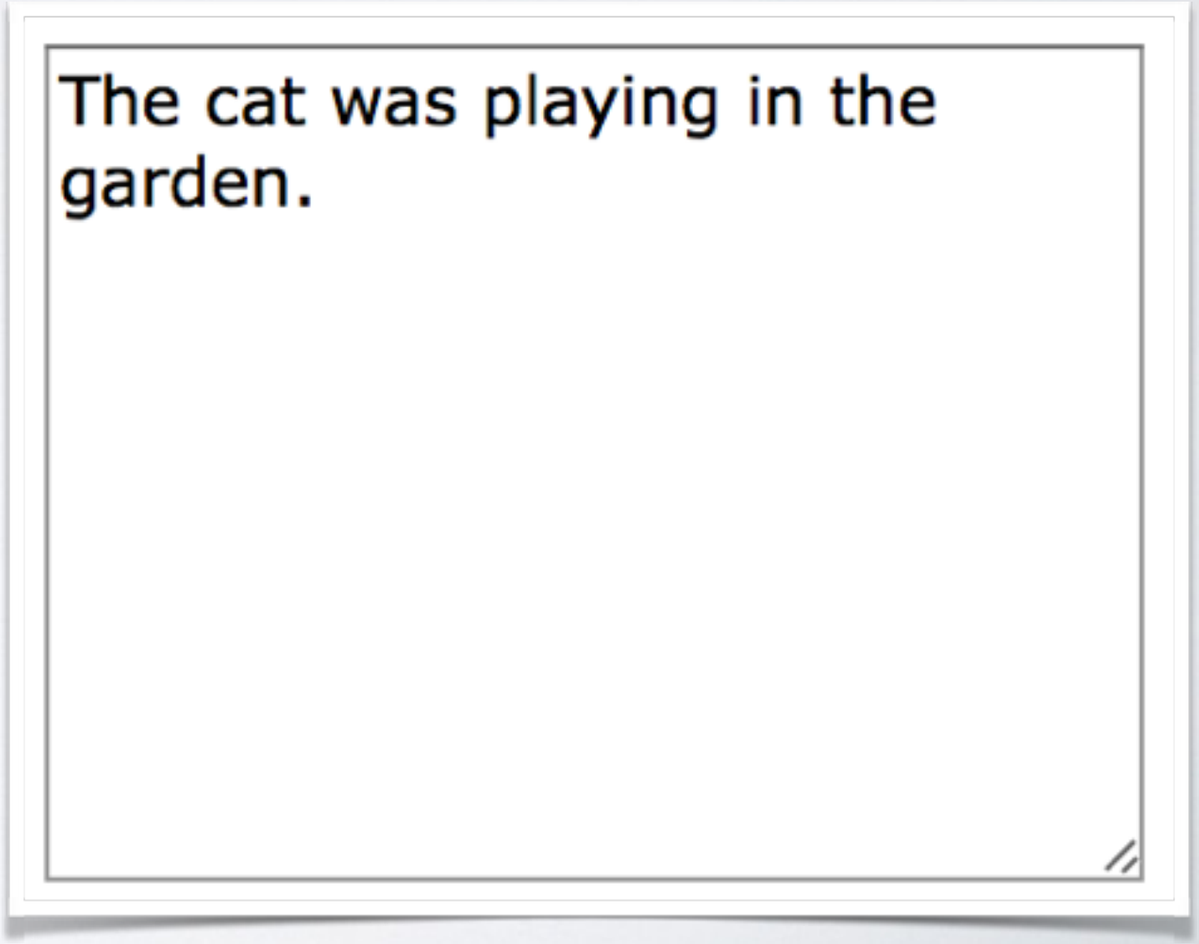
```
<select name="cars">
 <option value="volvo">Volvo</option>
 <option value="saab">Saab</option>
 <option value="fiat" selected>Fiat</option>
 <option value="audi">Audi</option>
</select>
```



Text Area

Enter in large amount of multi-line text.

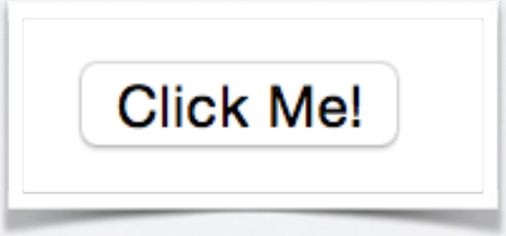
```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```



Generic Button

A non-submit mainly used for JavaScript coding.

```
<input type="button" onclick="alert('Hello World!')"
value="Click Me!" />
```



Input Attributes

`value:` specifies the initial/default value for an input field.

`readonly:` specifies that the input field cannot be changed.

`disabled:` specifies that the input field is disabled meaning they are un-usable and un-clickable and will not be submitted.

`size:` specifies the size (in characters) for the input field.

`maxlength:` specifies the maximum allowed length for the input field.

Python CGI

- A library to get the HTML form data.

  library *def.* an optional set of functions that can be loaded into memory and used.

```
import cgi

form = cgi.FieldStorage()
text1 = form.getvalue("text1")
text2 = form.getvalue("text2")
```

library

get the form data

get the data for the HTML element of the given name.

Making the Connections

How is data linked from HTML to Python?

form.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Form Example</title>
  <link rel="stylesheet" href="formstyle.css" type="text/css" />
</head>
<body>
  <h1>Form Example</h1>
  <form action="form.py">
    <div class="formin"> (a)
      <input type="text" name="text1" value="A textbox" />
    </div>
    <div class="formin"> (b)
      <input type="text" size="6" maxlength="10" name="text2" />
    </div>
    <div class="formin"> (c)
      <input type="submit" value="Go!" />
    </div>
  </form>
</body>
</html>
```

form.py

```
import cgi

form = cgi.FieldStorage()
text1 = form.getvalue("text1")
text2 = form.getvalue("text2")

# print HTTP/HTML headers
print """Content-type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html><head>
<title>A CGI Script</title>
</head><body>
"""

# print HTML body using form data
print "<p>In the first text box, you entered " + text1 + ".</p>"
print "<p>In the second text box, you entered " + text2 + ".</p>"
print "</body></html>"
```

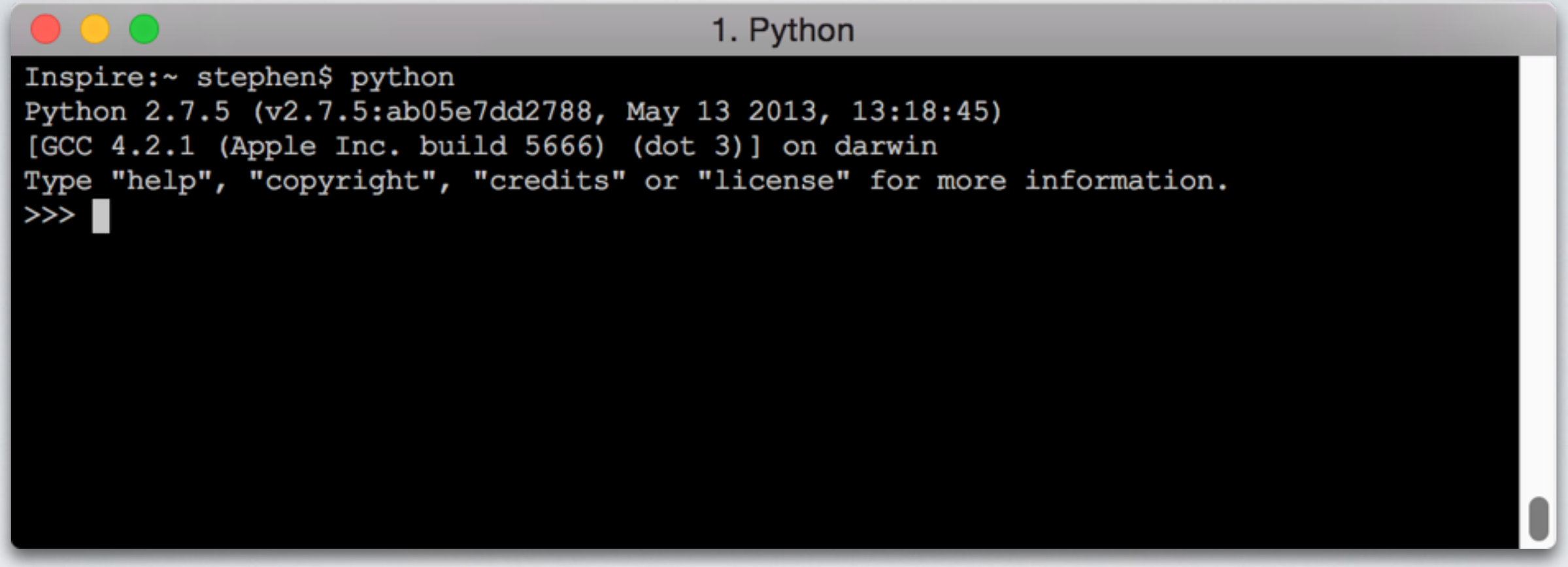
Through the **`action`** and **`name`** attributes of the form and input elements.

Form Data Conditions

3 conditions to check when programming dynamic HTML:

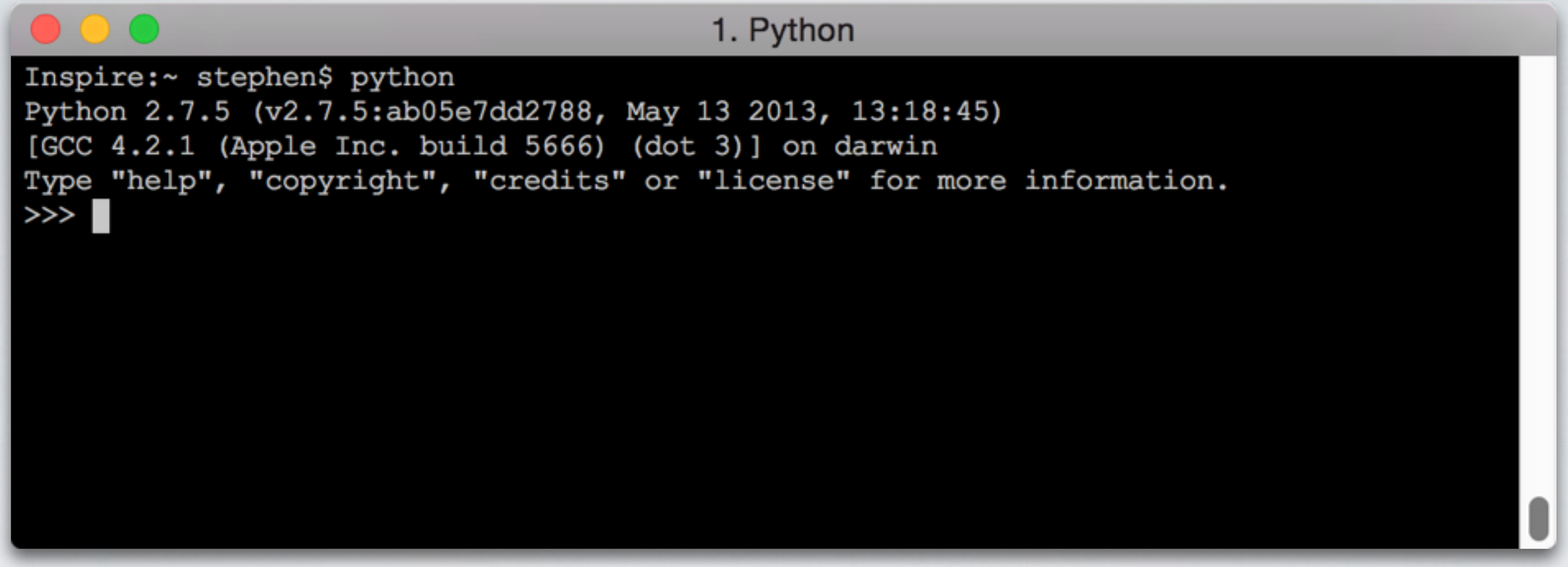
1. **No Form Input**
   - Browser is requesting the form for the first time (self-contained scripts, see *add.py example*).
2. **Invalid Form Input**
   - Some or all form data is invalid.
   - Send error back to browser so that data can be fixed.
3. **Valid Form Input**
   - All form data is OK, process, send results back to the browser.

Add Script Demo



Let us create a self-contained web script that can add 2 integers together from form input and output the sum. Only one python script is needed. No HTML or CSS files are needed.

Divide Script Demo



Take the add.py script and rename it to **divide.py**. Modify it so that instead of adding 2 integers in now divides 2 integers. Make sure that the output is a float and that you handle *division by zero* errors gracefully.

Summary

- Compared static HTML to dynamic HTML.
- Learnt about HTML form tags.
- Learnt about using Python as CGI scripts.
- Create simple forms to collect user data.
- Create scripts to process data and return a result.

Next Unit: look at advanced web programming.

QUESTIONS?