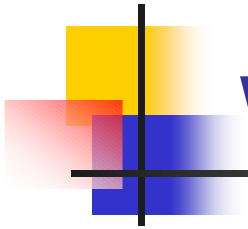
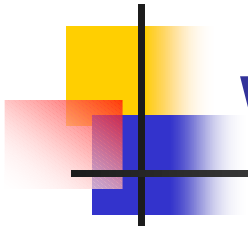


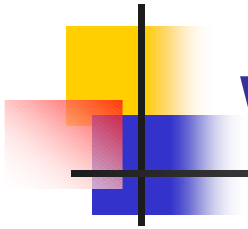
Webware for Python

- Developers:
  - Chuck Esterbrook
  - Jay Love
  - Tom Schwaller
  - Geoff Talvola
  - And many others have contributed patches
- http://webware.sourceforge.net/
- Mailing lists: webware-discuss and webware-devel
- Very helpful Wiki
- Birds of a Feather session 8:00 PM – 9:30 PM tonight!



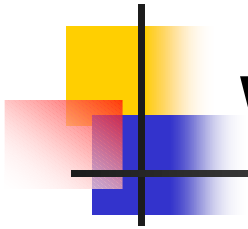
What is Webware?

- Python-oriented
- Object-oriented
- Cover common needs of web developers
- Modular architecture: components can easily be used together or independently
- Excellent documentation and examples
- Open source development and community
- Python-style license
- Cross-platform; works equally well on:
  - Unix in its many flavors
  - Windows NT/2000/XP



What is in Webware?

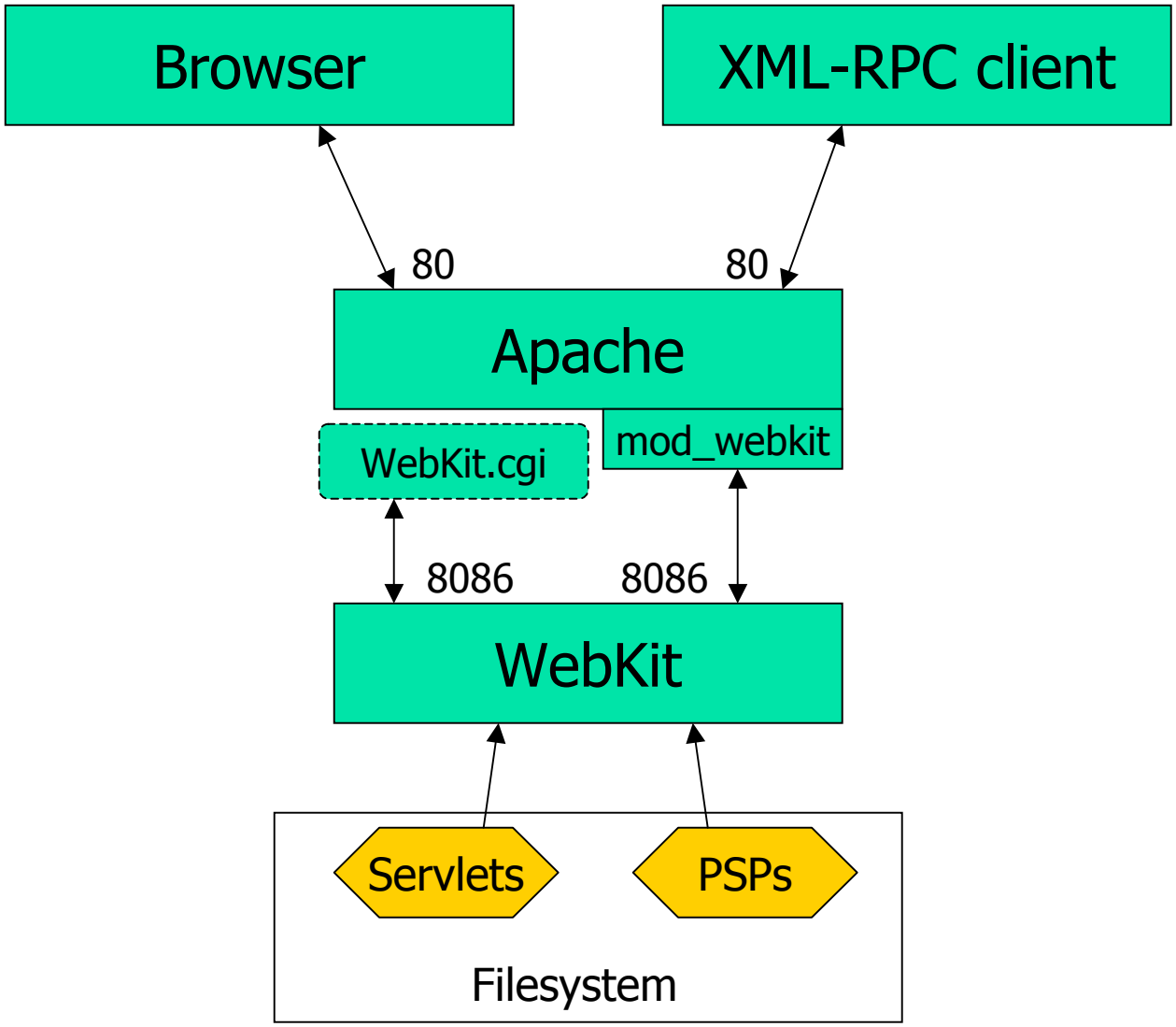
- The heart of Webware is WebKit
- We will also cover:
  - Python Server Pages (PSP)
  - TaskKit
  - MiddleKit
  - UserKit

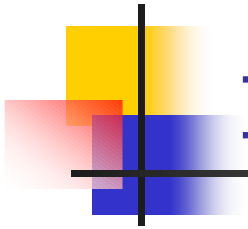


WebKit

- A fast, easy-to-use application server
- Multi-threading, not forking
  - Makes persistent data easier
  - Works well on Windows
- Stable and mature
- Used in several real-world, commercial projects
- Supports multiple styles of development:
  - Servlets
  - Python Server Pages

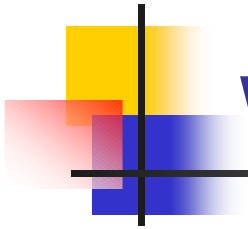
Architecture





Installing Webware

- Download
  - Latest official release can be downloaded from http://webware.sourceforge.net/
  - Or use CVS to pull in newer sources
- Install
  - Unpack the tarball, creating a Webware directory
  - Run **python install.py** in the Webware directory



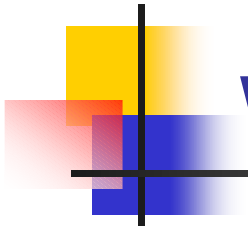
Working Directory

- You can run WebKit directly from the installation directory.
- But it's easy to create a separate working directory.
- Advantages:
  - Keeps configuration, logs, caches, servlets, etc. separate from the Webware directory
  - Lets you run multiple instances of WebKit without having to make multiple copies of Webware
  - Makes it easier to keep Webware up-to-date, since you don't have to modify it

Working Directory continued

- How to do it:
  - **python bin/MakeAppWorkDir.py /path/to/workdir**
- This creates this directory structure:

workdir/	**Cache/**		***used by Webware***
	Cans/		***???***
	Configs/	**Application.config**	***edit these to alter your configuration***
		AppServer.config	
	ErrorMsgs/		***Webware stores error messages here***
	Logs/		***Webware stores logs here***
	MyContext/		***Sample context is placed here; you can modify it to create your application***
	Sessions/		***Session data is stored here***
	AppServer		***Starts the AppServer on Unix***
	AppServer.bat		***Starts the AppServer on Windows***
	Launch.py		***Used by AppServer[.bat]***
	NTService.py		***Win NT/2000 Service version of AppServer***
	WebKit.cgi		***Install in your cgi-bin dir***
	OneShot.cgi		***Install in your cgi-bin dir to use One-Shot mode***



WebKit.cgi

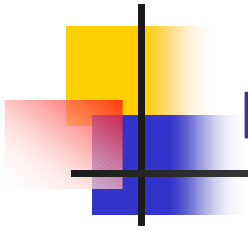
- Easy to install
- Should work with any web server that supports CGI
- To install:
  - Copy WebKit.cgi from your working directory (not from the Webware installation directory) to your web server's cgi-bin directory
  - On Windows, you will probably have to change the first line of WebKit.cgi from

    #! /usr/bin/env python

    to

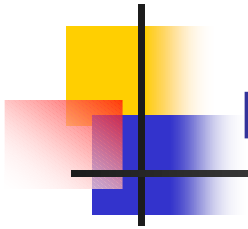
    #! C:\Python22\python.exe

    (or wherever Python is installed…)



mod_webkit

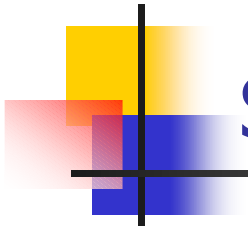
- Custom Apache module for Webware written in C
- Much faster than WebKit.cgi:
  - Does not have to start the Python interpreter on every request
- Located in **Webware/WebKit/Native/mod_webkit**
- On Unix:
  - use **make** and **make install**
- On Windows:
  - Download precompiled mod_webkit.dll from http://webware.sourceforge.net/MiscDownloads/
  - Place mod_webkit.dll into the **Apache/modules** directory



mod_webkit continued

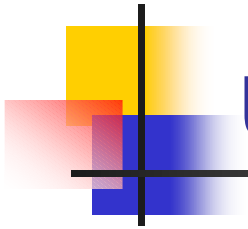
- Edit your Apache httpd.conf file:

```
# Load the mod_webkit module
# On windows you'd use mod_webkit.dll instead of mod_webkit.so
LoadModule webkit_module modules/mod_webkit.so
AddModule mod_webkit.c

# Include this if you want to send all .psp files to WebKit,
# even those that aren't found in a configured WebKit context.
AddType text/psp .psp
AddHandler psp-handler .psp

# This sends requests for /webkit/... to the appserver on port 8086.
<Location /webkit>
WKServer localhost 8086
SetHandler webkit-handler
</Location>
```



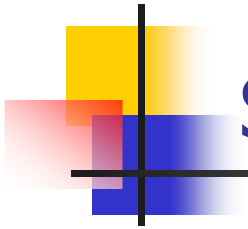
Starting the app server

- In your working directory, run:
  - Unix: **./AppServer**
  - Windows: **AppServer.bat**



Using the Example servlets and PSP's

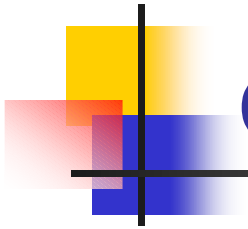
- To use the CGI adapter, surf to:
  - http://localhost/cgi-bin/WebKit.cgi
- To use the mod_webkit adapter, surf to:
  - http://localhost/webkit
- Experiment and enjoy!



Servlets

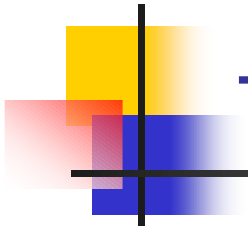
- A Python class located in a module of the same name
- Must inherit from **WebKit.Servlet** or one of its subclasses:
  - **WebKit.HTTPServlet**
  - **WebKit.Page**
- A common technique is to make your own subclass of **WebKit.Page** called **SitePage** which will contain:
  - Utility methods
  - Overrides of default behavior in **WebKit.Page**
- Simplest servlet:

```
from WebKit.Page import Page

class HelloWorld(Page):
    def writeContent(self):
        self.writeln('Hello, World!')
```



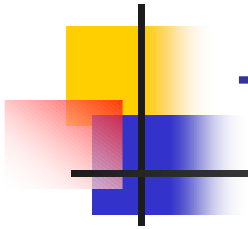
Contexts

- Servlets are located in Contexts
- A context is a Python package
  - Like a Python package, it contains an **__init__.py** module which:
    - Is imported before any servlets are executed
    - Is a good place to put global initialization code
    - If it contains a contextInitialize function, then contextInitialize(application, path_of_context) is called
- **Application.config** contains settings that map URL's to contexts
- Best to put non-servlet helper modules into a separate package, instead of putting them into the context package.



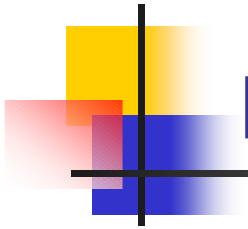
The Request-Response Cycle

- User initiates a request:
  - http://localhost/webkit/MyContext/MyServlet
- This activates the MyContext context, and the MyServlet servlet, based on settings in Application.config
  - Note: no extension was specified, even though the file is called MyServlet.py
  - There are several settings in Application.config that control the way extensions are processed
- An instance of the MyServlet class is pulled out of a pool of MyServlet instances, OR if the pool is empty then a new MyServlet instance is created.
- A Transaction object is created.
- These methods are called on the MyServlet instance:
  - Servlet.awake(transaction)
  - Servlet.respond(transaction)
  - Servlet.sleep(transaction)
- The MyServlet instance is returned to its pool of instances.



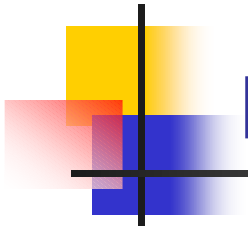
The Transaction Object

- Groups together several objects involved in processing a request:
  - Request: contains data received from the user
  - Response: contains the response headers and text
  - Servlet: processes the Request and returns the result in the Response
  - Session: contains server-side data indexed by a cookie
    - Can also use a variable embedded in the URL
  - Application: the global controller object
- You rarely use the transaction object directly



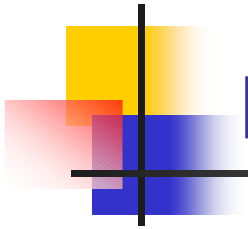
HTTPRequest

- Derived from generic Request base class
- Contains data sent by the browser:
  - GET and POST variables:
    - **.field(name, [default])**
    - **.hasField(name)**
    - **.fields()**
  - Cookies:
    - **.cookie(name, [default])**
    - **.hasCookie(name)**
    - **.cookies()**
  - If you don't care whether it's a field or cookie:
    - **.value(name, [default])**
    - **.hasValue(name)**
    - **.values()**
  - CGI environment variables
  - Various forms of the URL
  - Server-side paths
  - etc.



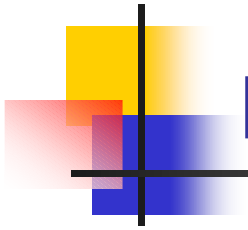
HTTPResponse

- Derived from generic Response base class
- Contains data returned to the browser
  - **.write(text)** – send text response to the browser
    - Normally all text is accumulated in a buffer, then sent all at once at the end of servlet processing
  - **.setHeader(name, value)** – set an HTTP header
  - **.flush()** – flush all headers and accumulated text; used for:
    - Streaming large files
    - Displaying partial results for slow servlets
  - **.sendRedirect(url)** – sets HTTP headers for a redirect



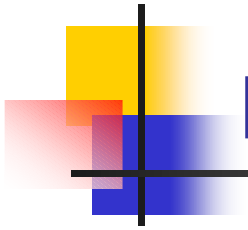
Page: Convenience Methods

- Access to the transaction and its objects:
  - **.transaction(), .reponse(), .request(), .session(), .application()**
- Writing response data:
  - **.write()** – equivalent to .response().write()
  - **.writeln()** – adds a newline at the end
- Utility methods:
  - **.htmlEncode()**
  - **.urlEncode()**
- Passing control to another servlet:
  - **.forward()**
  - **.includeURL()**
  - **.callMethodOfServlet()**
- Whatever else YOU decide to add to your SitePage



Page: Methods Called During A Request

- **.respond()** usually calls **.writeHTML()**
- Override **.writeHTML()** in your servlet if you want your servlet to provide the full output
- But, by default **.writeHTML()** invokes a convenient sequence of method calls:
  - **.writeDocType()** – override this if you don't want to use HTML 4.01 Transitional
  - **.writeln('<html>')**
  - **.writeHead()**
  - **.writeBody()**
  - **.writeln('</html>')**

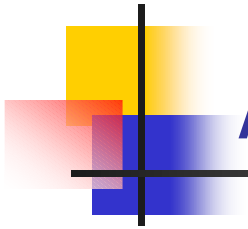


Page: .writeHead()

- **.writeHead()** calls:
  - **.write('<head>')**
  - **.writeHeadParts()** which itself calls:
    - **.writeTitle()**
      - Provide a **.title()** in your servlet that returns the title you want
      - Otherwise, the title will be the name of your servlet class
    - **.writeStyleSheet()** – override if you use stylesheets
  - **.write('</head>')**

Page: .writeBody()

- **.writeBody()** calls:
  - **.write('<body %s>' % self.htBodyArgs())**
    - override **.htBodyArgs()** if you need to provide arguments to the <body> tag
  - **.writeBodyParts()** which itself calls:
    - **.writeContent()**
      - usually this is what you'll override in your servlets or SitePage
  - **.write('</body>')**



Actions

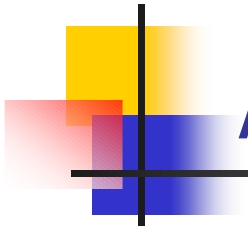
- Actions are used to associate different form submit buttons with different servlet methods
- To use actions:
  - Add submit buttons like this to a form:

    <input name=_action_add type=submit value="Add Widget">
  - Provide a **.actions()** method which returns list of method names:

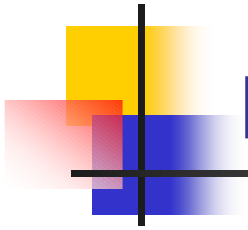
```
def actions(self):
    return ['add', 'delete']
```

  - **.respond()** checks for a field **_action_ACTIONNAME** where **ACTIONNAME** is in the list returned by **.actions()**
    - If such a field is found, then **.handleAction()** is called instead of **.writeHTML()**

Actions continued



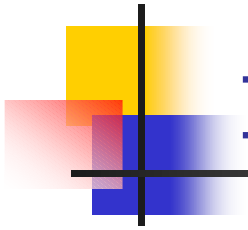
- **.handleAction()** calls:
  - **.preAction(ACTIONNAME)** which itself calls:
    - **.writeDocType()**
    - **.writeln('<html>')**
    - **.writeHead()**
  - **.ACTIONNAME()**
  - **.postAction(ACTIONNAME)** which itself calls:
    - **.writeln('</html>')**
- In other words, your action method is called instead of **.writeContent()**
- Of course, you don't have to use actions at all; you can simply write code in your **writeContent** that examines the HTTPResponse object and acts accordingly.



Forwarding

- **self.forward('AnotherServlet')**
  - Analogous to a redirect that happens entirely within WebKit
  - Bundles up the current Request into a new Transaction
  - Passes that transaction through the normal Request-Response cycle with the indicated servlet
  - When that servlet is done, control returns to the calling servlet, but all response text and headers from the calling servlet are discarded
  - Useful for implementing a "controller" servlet that examines the request and passes it on to another servlet for processing
  - Until recently, you had to write:

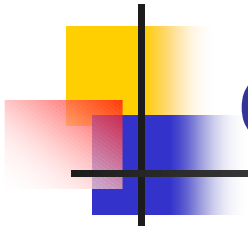
```
self.application().forward(self.transaction(), 'AnotherServlet')
```



Including

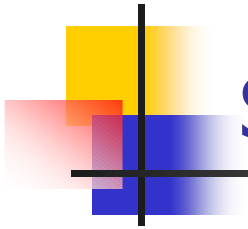
- **self.includeURL('AnotherServlet')**
  - Similar to **.forward()**, except that the output from the called servlet is *included* in the response, instead of *replacing* the response.
  - Until recently, you had to write:

```
self.application().includeURL(self.transaction(), 'AnotherServlet')
```



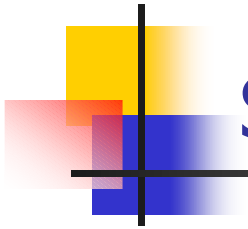
Calling Servlet Methods

- **self.callMethodOfServlet(‘AnotherServlet’, ‘method’, arg1, arg2, …)**
  - Instantiates the indicated servlet
  - Calls **servlet.awake()**
  - Calls the indicated method with the indicated args
  - Calls **servlet.sleep()**
  - Returns the return value of the method call back to the calling servlet
  - Example: suppose you have a table-of-contents servlet that needs to fetch the title of other servlets by calling the **.title()** method on those servlets:
    - **title = self.callMethodOfServlet(servletName, ‘title’)**



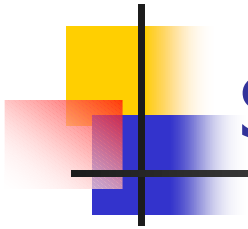
Sessions

- Store user-specific data that must persist from one request to the next
- Sessions expire after some number of minutes of inactivity
  - Controlled using **SessionTimeout** config variable
- The usual interface:
  - .value(name, [default])
  - .hasValue(name)
  - .values()
  - .setValue(name, value)



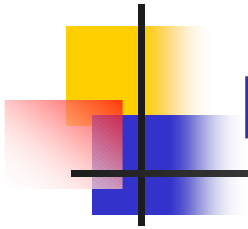
Session Stores

- Three options for the **SessionStore** config variable:
  - Memory – all sessions are kept in memory
  - Dynamic – recently used sessions are kept in memory, but sessions that haven't been used in a while are pickled to disk and removed from memory
    - This is the default, and it is recommended.
  - File – sessions are pickled to disk and unpickled from disk on every request and are not stored in memory at all.
    - Not recommended.
- All sessions are pickled to disk when the appserver is stopped, and unpickled when the appserver starts.
  - You can restart the appserver without losing sessions.



Session Options

- Sessions are keyed by a random session ID
- By default, the session ID is stored in a cookie
- Alternative: set **UseAutomaticPathSessions** to 1
  - The session ID is automatically embedded as a component of the URL
  - Cookies not required
  - But: URLs become much longer and uglier
- No way (yet) to have WebKit choose the appropriate strategy based on whether the browser supports cookies



PSP: Python Server Pages

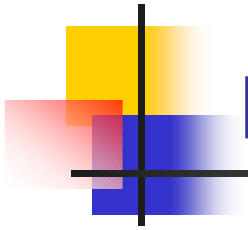
- Mingle Python and HTML in the style of JSP or ASP
- Include code using **<% ... %>**
- Include evaluated expressions using **<%= ... %>**
- Begin a block by ending code with a colon:

  <%for I in range(10):%>
- End a block using the special tag:

  <%end%>
- When the user requests a PSP:
  - It is automatically compiled into a servlet class derived from **WebKit.Page**
  - The body of your PSP is translated into a **writeHTML()** method

PSP Example

```
<%
def isprime(number):
    if number == 2:
        return 1
    if number <= 1:
        return 0
    for i in range(2, number/2):
        for j in range(2, i+1):
            if i*j == number:
                return 0
    return 1
%>

<p>Here are some numbers, and whether or not they are prime:
<p>

<%for i in range(1, 101):%>
    <%if isprime(i):%>
        <font color=red><%=i%> is prime!</font>
    <%end%><%else:%>
        <%=i%> is not prime.
    <%end%>
    <br>
<%end%>
```



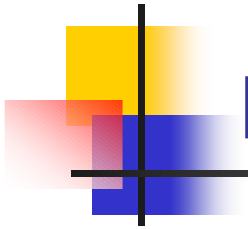
PSP Directives

- **<%@ page imports="module, package.module, package:module" %>**
  - equivalent to at module level:
    - **import module**
    - **import package.module**
    - **from package import module**
- **<%@ page extends="MyPSPBaseClass" %>**
  - makes the generated servlet derive from the specified class
- **<%@ page method="writeContent" %>**
  - makes the body of your PSP be placed into a **writeContent** method instead of the **writeHTML** method.
- **<%@ page indentType="braces" %>**
  - Ignores indentation; uses braces for grouping

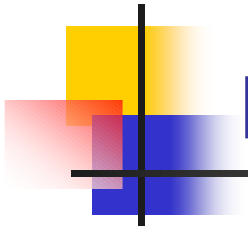
PSP: Braces Example

```
<%@page indentType="braces"%>
<%
def isprime(number): {
   if number == 2: {
      return 1
   } if number <= 1: {
      return 0
   } for i in range(2, number/2+1): {
      for j in range(2, i+1): {
         if i*j == number: {
            return 0
         }
      }
   }
   return 1
}
%>
<p>Here are some numbers, and whether or not they are prime:
<p>
<%
for i in range(1, 101): {
   if isprime(i): { %>
      <font color=red><%=i%> is prime!</font>
   <%} else: {%>
      <%=i%> is not prime.
   <%}%>
   <br>
<%}%>
```

PSP: Four Ways To Include



- **<%@ include file="myinclude.psp"%>**
  - Includes the specified file at compile time and parses it for PSP content, like #include in C
  - If included file's contents changes, you must restart the app server to pick up the change
- **<psp:include path="myinclude">**
  - Equivalent to **self.includeURL('myinclude')**
  - Changes to the included file's contents are reflected immediately
- **<psp:insert file="myinclude.html">**
  - File is included verbatim in the output. No PSP parsing.
  - File is read from disk for every request, so changes to the included file's contents are reflected immediately
- **<psp:insert file="myinclude.html" static="1">**
  - Includes the specified file at compile time verbatim, without parsing for PSP content.
  - If included file's contents changes, you must restart the app server to pick up the change



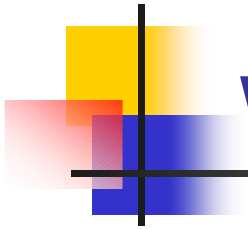
PSP: Methods

- Adding methods to a PSP servlet with the psp:method directive:

```
<psp:method name="add" params="a,b">
return a + b
</psp:method>

100 + 200 = <%=self.add(100, 200)%>
```

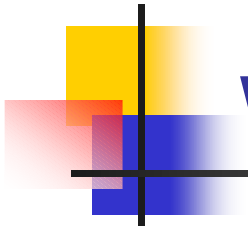
- Here's a slightly less contrived example:

```
<%@ page method="writeContent" %>

<psp:method name="title">
return 'Prime Numbers'
</psp:method>
```



Web Services: XML-RPC

- Turn your Webware site into a “web service”
- Write a servlet derived from **XMLRPCServlet**
  - Define **exposedMethods()** method that lists the methods you want to expose through XML-RPC
  - Write your methods

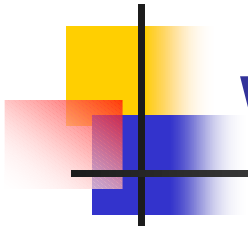


Web Services: XML-RPC Servlet Example

```
from WebKit.XMLRPCServlet import XMLRPCServlet

class XMLRPCExample(XMLRPCServlet):
    def exposedMethods(self):
        return [‘multiply’, ‘add’]
    def multiply(self, x, y):
        return x*y
    def add(self, x, y):
        return x+y
```

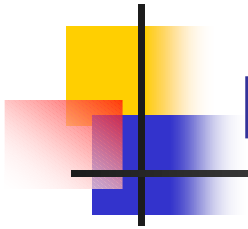
Web Services: XML-RPC Client Example

```
import xmlrpclib

servlet = xmlrpclib.Server(
      'http://localhost/webkit/Examples/XMLRPCExample')

print servlet.add('foo', 'bar')
print servlet.multiply('foo', 3)
Print servlet.add('foo', 3)  # This raises an exception
```



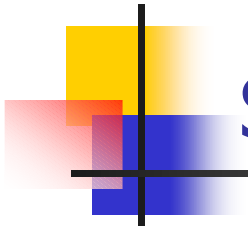
Web Services: XML-RPC continued

- Exceptions are propagated as XML-RPC Faults
  - Configuration setting **IncludeTracebackInXMLRPCFault** controls whether or not the full traceback is included in the Fault
- Easy to customize XML-RPC Servlet behavior
  - Just override **call()** in a subclass
  - Examples:
    - Suppose you want an authentication token or session ID to be the first parameter of every method
    - Rather than add that parameter to every method, just write a custom **call()** method



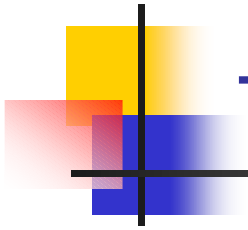
PickleRPC

- Brand-new in Webware CVS
- Uses Python's pickle format instead of xmlrpc format
- Advantages:
  - Works correctly with all Python types that can be pickled, including longs, None, mx.DateTime, recursive objects, etc.
  - Faster (?)
- Disadvantages:
  - Python-specific
  - Security holes (may be addressed soon)



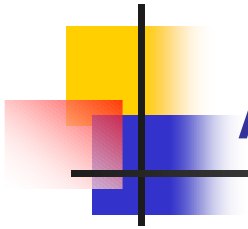
ShutDown handlers

- As we learned before, the **contextInitialize(application, path)** function in an **__init__.py** in a context is a good place to put global initialization code
- Where do you put global finalization code?
- Answer:
  - Register a shutdown handler function with **application.addShutDownHandler(func)**
  - On shutdown, all functions that have been registered using **addShutDownHandler** get called in the order they were added.
- New in CVS



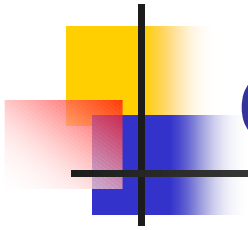
Tracebacks

- If an unhandled exception occurs in a servlet:
  - Application.config settings:
    - If **ShowDebugInfoOnErrors** = 1, an HTML version of the traceback will be shown to the user; otherwise, a short generic error message is shown.
    - You can configure WebKit so that it sends the traceback by email: **EmailErrors**, **ErrorEmailServer**, **ErrorEmailHeaders**
    - Include "fancy" traceback using **IncludeFancyTraceback** and **FancyTracebackContext**
- Your users will NOT report tracebacks, so set up emailing of fancy tracebacks!



Admin pages

- Password-protected
- Detailed activity log
- Detailed error log
- View configuration settings
  - Application.config
  - AppServer.config
- View plug-ins
- View servlet cache
- Application Control
  - Shut down the app server
  - Clear the servlet cache
  - Reload selected modules
- My opinion: probably NOT a good idea to enable the admin pages in a production site due to security concerns

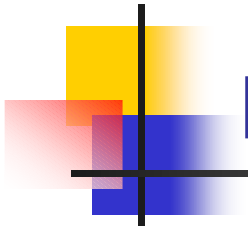


One-Shot

- Webware automatically reloads servlets whose source code has changed on disk
- Webware does NOT reload dependencies when they change
- Solution: **OneShot.cgi**
  - CGI script that fires up the app server, handles one request, and shuts down
  - Very useful for debugging if you have a fast machine and are not using any libraries that take a long time to load
  - Otherwise, can be unbearably slow
- Alternatives:
  - Custom **WebKit.cgi** that restarts the app server only if files have changed; see the Wiki
- Put a restart icon on your desktop. Windows example:

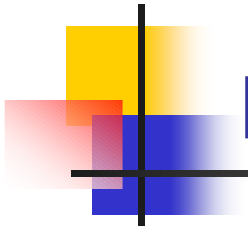
  net stop WebKit

  net start WebKit



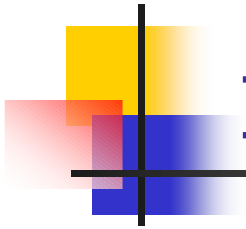
Deployment issues: Unix

- **WebKit/webkit**
  - Unix shell script launching WebKit at boot time using the standard “init” mechanisms
  - See the WebKit Install Guide and Wiki for hints
- **Monitor.py**
  - This starts up WebKit and monitors its health, restarting it if necessary.
  - I’ve never used this one



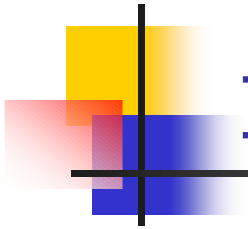
Deployment issues: Windows NT/2000

- Installing as a Service
  - Run **python NTService.py install** in your working dir
  - This creates a service called **WebKit App Server** with a short name of **WebKit**
  - Use the Services Control Panel to configure a user account and a startup policy (manual or automatic)
- Controlling the service
  - Use the Services Control Panel
  - From the command-line:
    - **net start WebKit**
    - **net stop WebKit**
- Removing the service
  - Stop the service
  - Run **python NTService.py remove**
- "Secret" AppServer.config setting: **NTServiceLogFilename** (will change in the future)



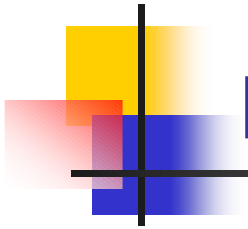
IIS: wkcgi.exe

- CGI adapter written in C for greater speed
- If you have to use IIS, this is your best option
- Not as fast as Apache with mod_webkit
- Download compiled version from http://webware.sourceforge.net/MiscDownloads/
- Connects to localhost:8086 by default
  - If you need to connect elsewhere, place a webkit.cfg file in the same directory
  - See Webware/WebKit/Native/wkcgi/webkit.cfg for a sample



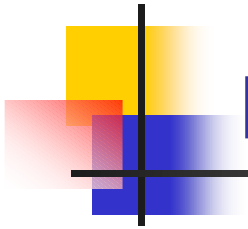
IIS: wkISAPI

- Experimental ISAPI module for IIS that could result in speed equal to Apache with mod_webkit
- Needs testing
- Rumored to have memory leaks



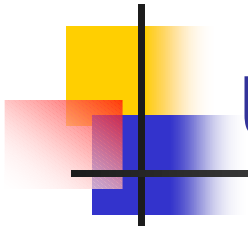
MiddleKit

- Object-Relational mapper
- Supports MySQL and MS SQL Server.
  - PostgreSQL support soon?
- Can be used anywhere, not just WebKit applications.
- Write an object model in a Comma-Separated Values (CSV) file using a spreadsheet
  - Inheritance is supported
  - Numbers, strings, enums, dates/times, object references, lists of objects (actually sets of objects)
- Compile the object model
  - This generates Python classes for each of your objects that contain accessor methods for all fields
  - Also, an empty derived class is provided where you can add your own methods
  - And, a SQL script is generated that you can run to create the tables



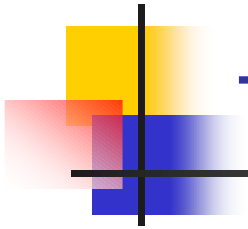
MiddleKit continued

- In your application code:
  - Create a singleton instance of SQLObjectStore pointing it to your SQL Database and your object model CSV file
  - Use store.fetchObjectsOfClass() to fetch objects from the store as needed
  - Create objects using their constructor
  - Modify the objects using the accessor methods that were generated for you
  - Add objects to the store using store.addObject()
  - Save changes to the database using store.saveChanges()
  - Delete objects using store.deleteObject()
  - See the MiddleKit documentation for all the details



UserKit

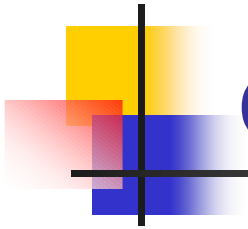
- Basic framework for user and role management
- Pre-alpha status; needs much more work



TaskKit

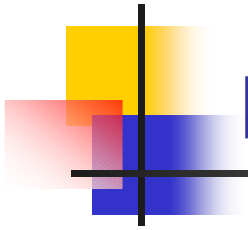
- Useful framework for scheduling periodic tasks
- Can be used outside of WebKit
- Example:

```
from TaskKit.Task import Task
from TaskKit.Scheduler import Scheduler

class MyTask(Task):
    def run(self):
        # Do something useful...

scheduler = Scheduler()
scheduler.start()
scheduler.addPeriodicAction(time() + 60*5, 60*5, MyTask(),
    'MyTask')
```



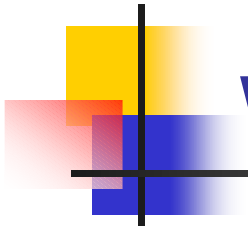
Cheetah

- http://www.cheetahtemplate.org/
- A Python-powered template engine and code generator
- Integrates tightly with Webware
- Can also be used as a standalone utility or combined with other tools
- Compared with PSP:
  - Much more designer-friendly
  - Perhaps less programmer-friendly?
- Paper on Cheetah being presented from 3:30-5:00 PM today



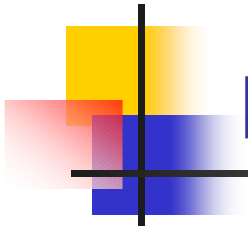
FunFormKit

- http://colorstudy.net/software/funformkit/
- A package for Webware that does:
  - Form validation
  - Value conversion
  - HTML generation
  - Re-querying on invalid input
  - Compound HTML widgets (for example a Date widget)
- LGPL license



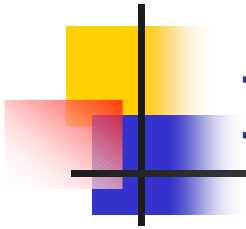
Who's using Webware?

- Public sites:
  - http://foreclosures.lycos.com/ - searchable database of foreclosure property
  - http://www.electronicappraiser.com/ - online home valuations
  - http://www.vorbis.com/ - home page for ogg vorbis audio encoding technology
- Private sites – intranets and extranets
  - Parlance Corporation: reporting and administrative capabilities for their customers
  - HFD: The Monkey, a content management system
  - Juhe: a membership management system for the Austrian Youth Hostel Association
  - Several others listed in the Wiki



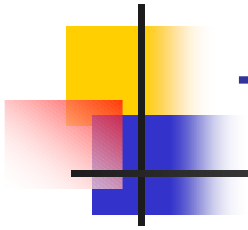
Future Plans

- Releases:
  - New release every 2 months
  - Next release 0.7 in 2nd half of February
- Planned features (partial list):
  - Comprehensive test suite
  - Improve documentation
    - Some features are undocumented
    - Install guide needs to be updated
  - PostgreSQL support in MiddleKit
  - Built-in HTTP server
  - Multi-application support
  - Distutils support



I Want To Contribute!

- See the Wiki for ideas on areas where we could use help
- Contribute patches on SourceForge
- Write a module for use with Webware
  - Could be useable independent of Webware (like Cheetah)
  - Could be Webware-specific (like FunFormKit)
    - Give it a “Kit” suffix
  - If it needs to hook into WebKit, make it a “Plug-In”
    - See WebKit/PlugIn.py for details
    - PSP is an example of a plug-in that happens to be included with Webware
- Please follow the Webware Style Guidelines
  - See the documentation



That’s All!

- Any questions?