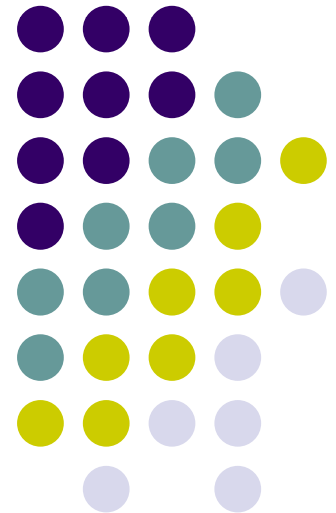
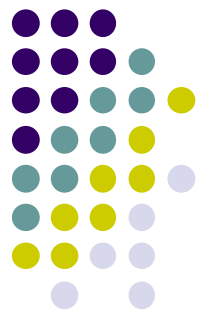


CSI2441: Applications Development

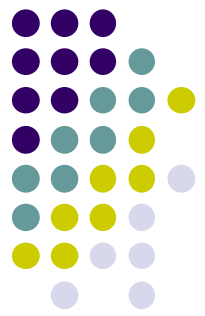
Lecture 4 *Designing and Writing Applications*





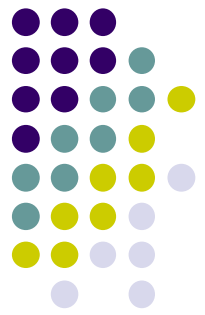
Objectives

- Plan the mainline logic for a complete program
- Describe typical housekeeping tasks
- Describe tasks typically performed in the main loop of a program
- Describe tasks performed in the end-of-job module
- Understand the need for good program design



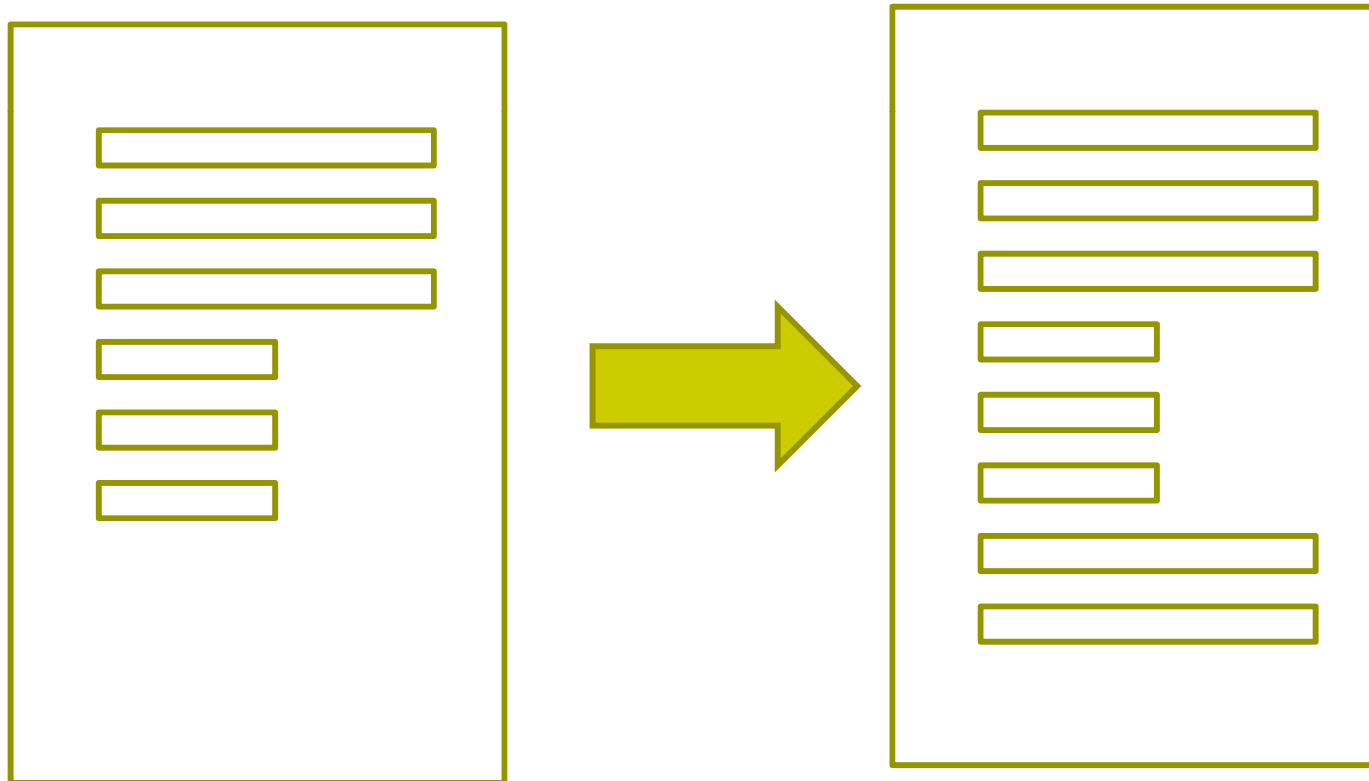
Objectives (continued)

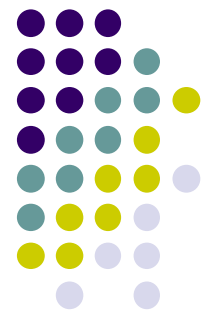
- Appreciated the advantages of storing program components in separate files
- Select superior variable and module names
- Design clear module statements
- Understand the need for maintaining good programming habits



Understanding the Mainline Logical Flow Through a Program

- Understand what the goals are
 - Ask the user to clarify if necessary





Understanding the Mainline Logical Flow Through a Program (continued)

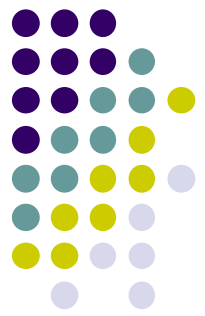
- Ensure you have all the data required to produce the desired output
- As stated last week, do your forms and form fields match what is being stored and returned from the database

INVENTORY FILE DESCRIPTION

File name: INVENTORY

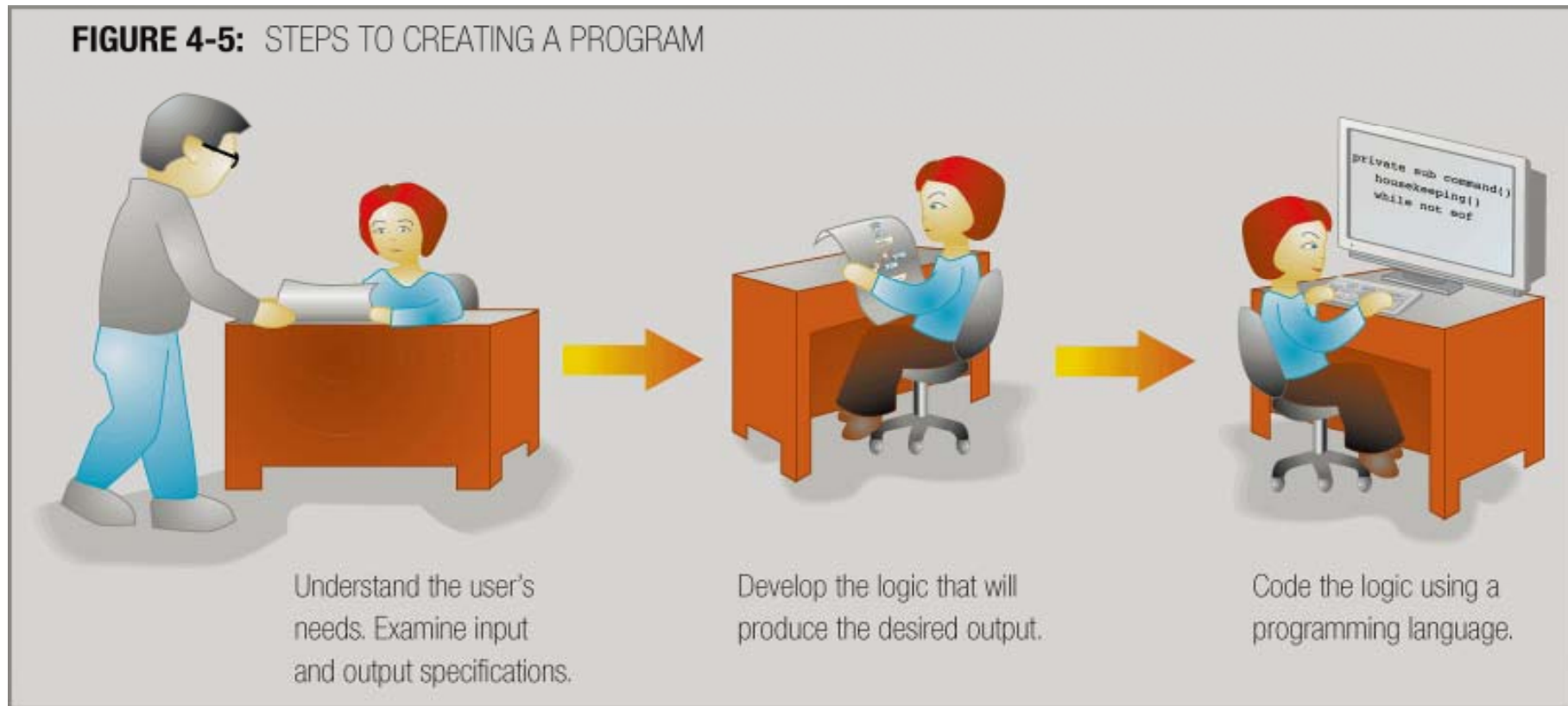
FIELD DESCRIPTION	DATA TYPE	COMMENTS
Item name	Character	15 bytes
Price	Numeric	2 decimal places
Cost	Numeric	2 decimal places
Quantity in stock	Numeric	0 decimal places

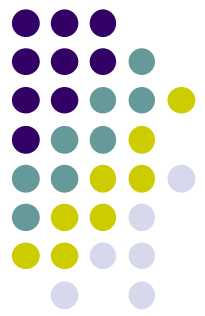
cotton shirt	01995	01457	2500
wool scarf	01450	01125	0060
silk blouse	16500	04850	0525
cotton shorts	01750	01420	1500



Understanding the Mainline Logical Flow Through a Program (continued)

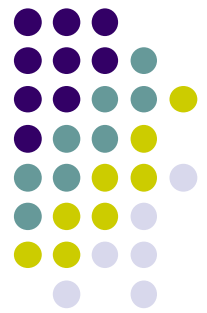
- Understand the big picture first



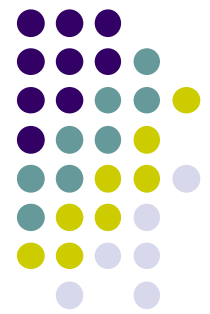


Understanding the Need for Good Program Design

- Good design is:
 - Critical for very large programs
 - Needed to guarantee that components work together properly
- Well-designed program modules should work:
 - As stand-alone modules
 - As part of larger systems

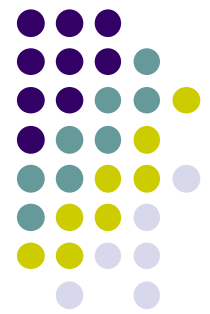


The Application Environment



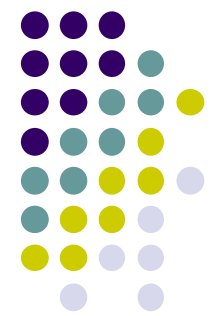
Environment Selection

- Developers cannot always assume they will be using the same languages and environments on all jobs
- You should choose the environment that suites the task, not the environment which suites the developers
- Issues include
 - What environment will your web app run on? (linux, Windows, Mac)
 - What language will you use?
 - What database system?
 - What are the hosting cost implications? Open source solutions such as php/mysql are cheaper to host online than proprietary solutions, such as .Net apps / ms sql server



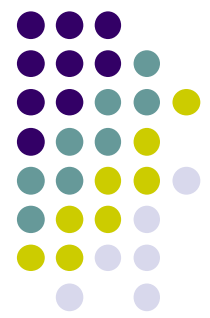
Hosting Services

- Web applications can be hosted in-house within a company
- They can also be hosted with an external providers
- Along with hosting comes the cost of domain names
- There are a huge selection of hosting services in Australia and across the world
- Some hosts provide all possible solutions (languages, databases, operating systems) while others specialise
- If a client already has a host, then as developers you may be locked in to the environments and versions supported by that host
- If not, the client needs to know the costs and issues involved with web application hosting so that they can be involved in the selection process
- While hosting services do not necessarily need to be selected prior to deployment, it does help



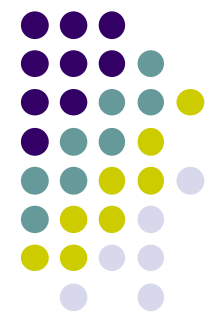
Internal Hosting

- Internal hosting means that a company should have sufficient internal technical support to keep web and database servers up and running
- Internally hosted web applications can either be internally accessible only, or be hosted internally be accessible from the public web
- Most large web applications separate public and internal access and then restrict this through user accounts and domain restrictions (ie only local in-house ip addresses can access certain parts of the site)
- Internal hosting gives more control and privacy to a company and its systems, but brings with it the requirement for support and management of system uptime
- Leased lines, fixed ip addresses and support staff can be expensive



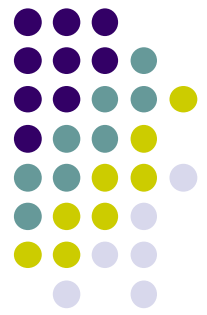
External Hosting

- External hosting reduces in-house technical support requirements
- Most large hosting services will provide Service Level Agreements meaning their systems will be 'up' a certain amount of time (say 99%)
- They can also provide load balancing for heavy use periods
- Obviously, application errors are not covered by their support – they provide the environment in which the application runs
- Modern hosting services have user-friendly interfaces for managing, configuring, uploading and backing up application code and data
- Hosting services typically now include payment gateways, https and shopping cart functionality out-of-the-box
- Perhaps the main drawback of external hosting is a lack of control of the environment and your application

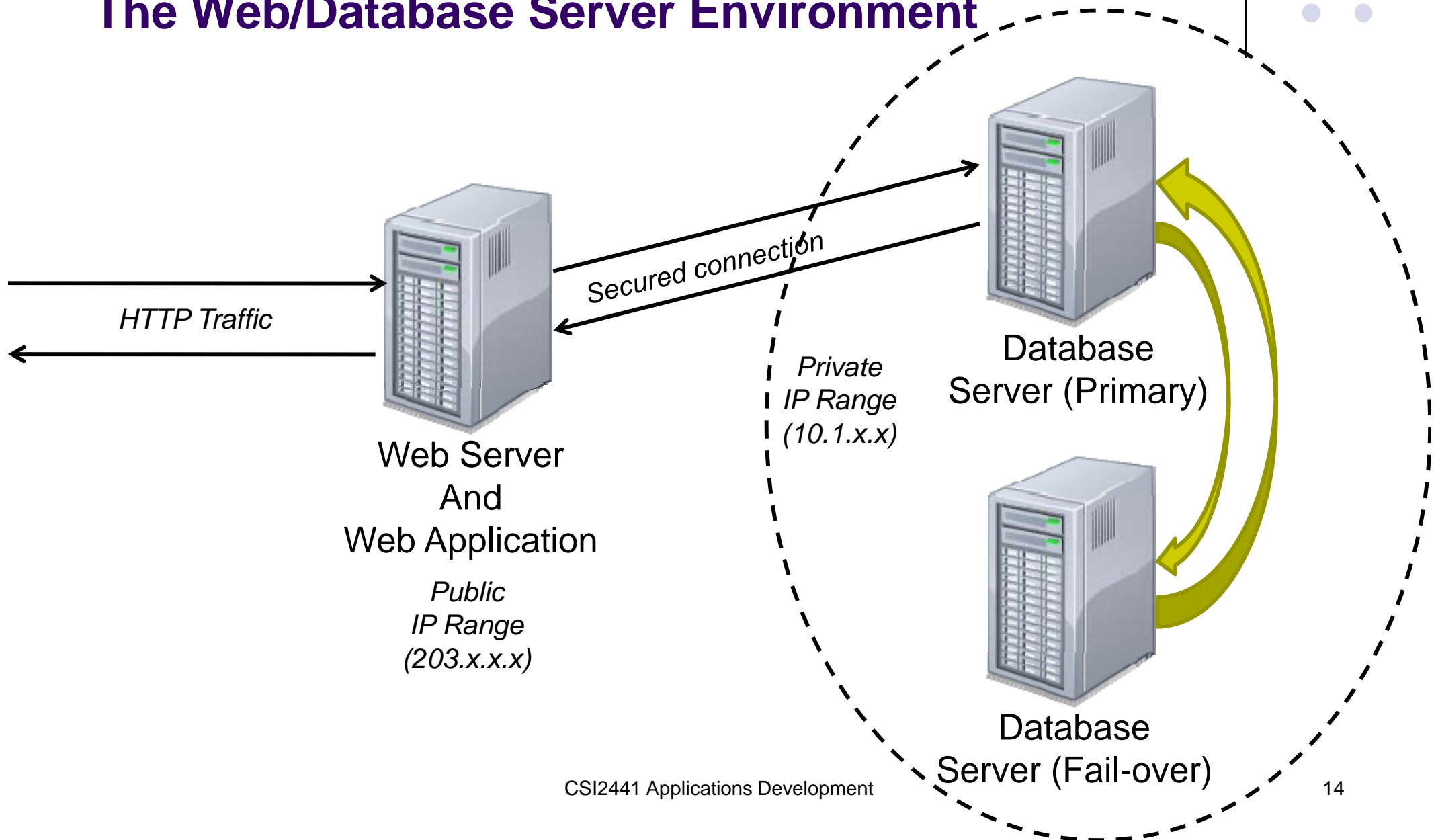


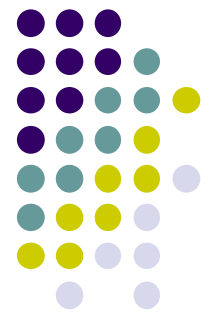
RDBMS Selection and Issues

- While the code in your application process the business logic, the business data is typically stored in a relational database management system
- Mainstream solutions include MySQL, MS Sql Server, Oracle, IBM's DB2, Postgres etc
- Some, like Postgres and MySQL are basically 'free', while Oracle and DB2 are 'big iron' and expensive
- Many require multi-processor machines, and are often licensed on a per-processor basis (ie 4 processors = 4 licenses)
- For mission critical applications, the database server needs to be mirrored or have a fail-over capacity (ie a exact duplicate machine which takes over instantly if the primary server fails)
- Databases should NEVER be facing the public web



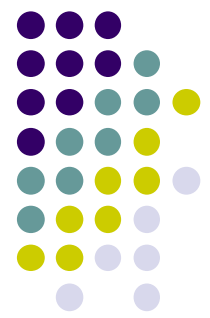
The Web/Database Server Environment





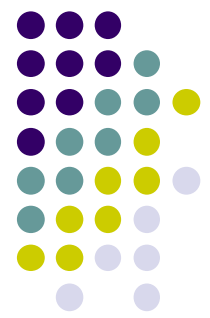
The Web/Database Server Environment

- In the previous slide, the database servers sit inside a local-only network topology
- The public-facing web server and application can access the database through a secured, port locked connection
- Firewalls can protect the web server, while proxies and firewalls can protect the internal network
- The public web facing side of the environment is called the DeMilitarisedZone (DMZ) – ie there is less control and order
- The public facing machine should be as secure as possible and run only what it needs to service is function – ie the operating system, web server and scripting environment for the application
- Unless otherwise required, all ports except port 80 (the HTTP port) should be locked off



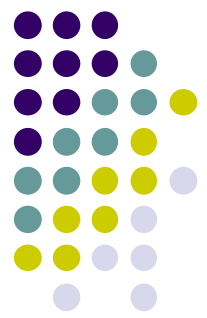
Test Environments

- Where possible, your development environment should mimic your production environment
- This means that changes made in the test environment should be more predictably reflected once moved to the production environment
- Especially critical is that both test and production environments have the same versions of;
 - Scripting language
 - Web server
 - Database server
 - User/security privileges



Code Libraries and Includes

- Just as you can store functionality in modules, functions or procedures, you can also store these structures in files external to your mainline logic
- Instead of having one (or more) huge script files, you place all your 'common' and reusable code elements into external files, which can be read in as needed
- Common types of code libraries can include;
 - Menu systems
 - Database connections
 - User authentication/logout
 - Usage logging



Include Example

- In this example, we create a menu in one page
- We call it in all pages that then need a menu included in it
- If we need to alter the menu system, we open one file and make our changes
- All files to 'include' that menu file see the change next time they load

menu.php Code:

```
<html>
<body>
<a href="http://www.example.com/index.php">Home</a> -
<a href="http://www.example.com/about.php">About Us</a> -
<a href="http://www.example.com/links.php">Links</a> -
<a href="http://www.example.com/contact.php">Contact Us</a> <br />
```

Save the above file as "menu.php". Now create a new file, "index.php" in the same directory as "menu.php". Here we will take advantage of the *include* command to add our common menu.

index.php Code:

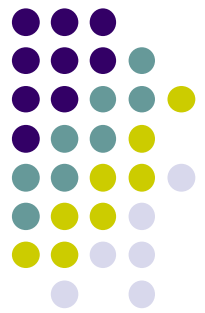
```
<?php include("menu.php"); ?>
<p>This is my home page that uses a common menu to save me time when I add
new pages to my website!</p>
</body>
</html>
```

Display:

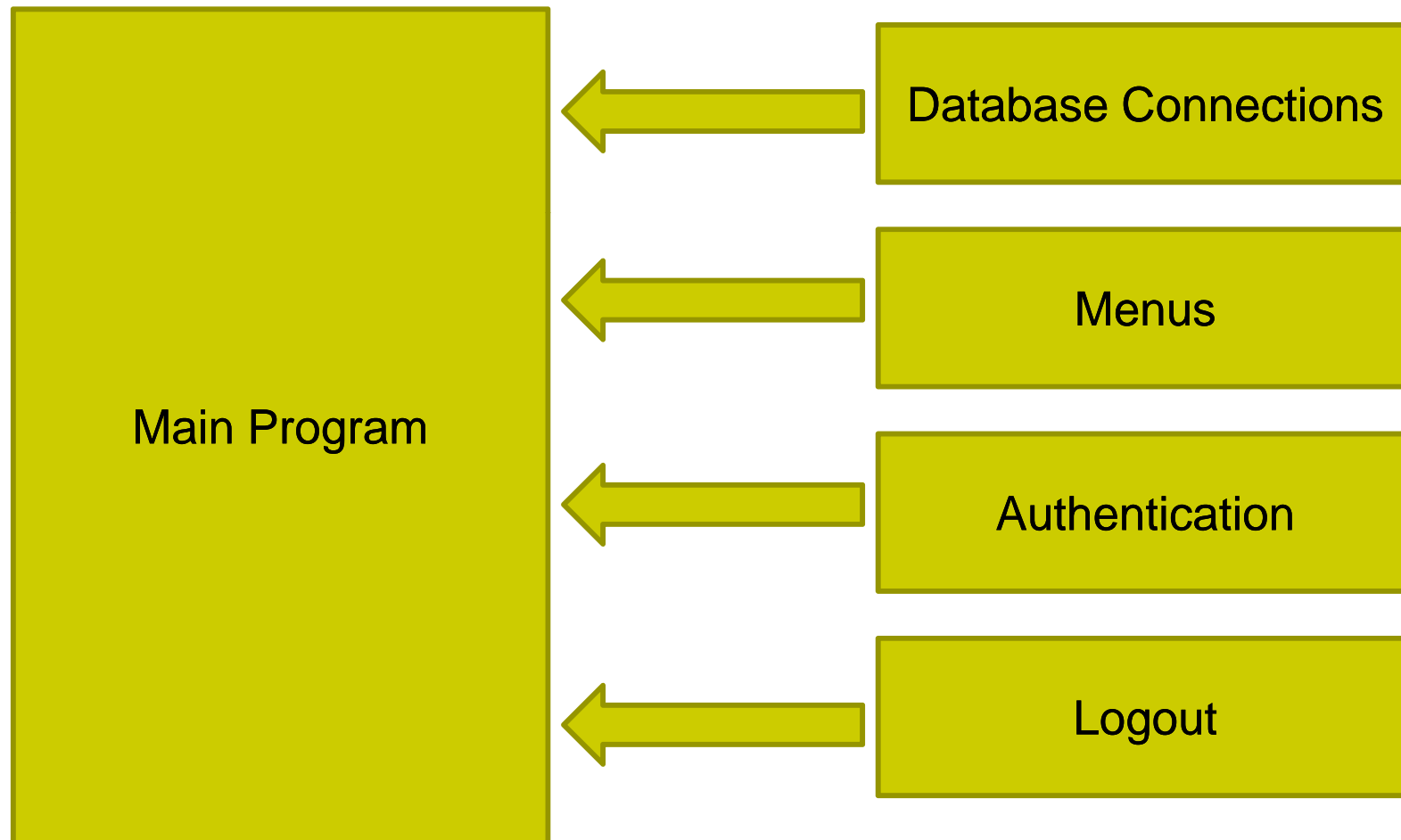
[Home](#) - [About Us](#) - [Links](#) - [Contact Us](#)

This is my home page that uses a common menu to save me time when I add new pages to my website!

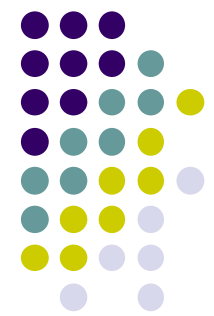
<http://www.tizag.com/phpT/include.php>



Include Cont....

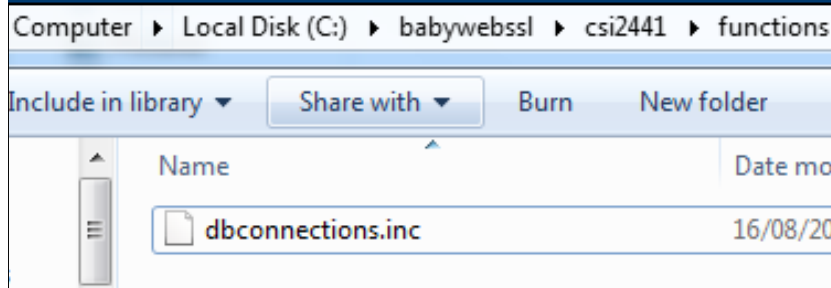
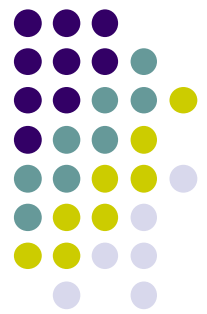


<http://www.tizag.com/phpT/include.php>

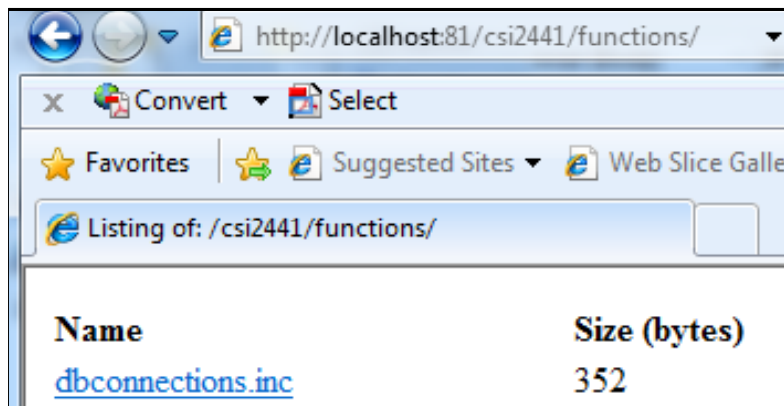


Storing Program Components in Separate Files (issues)

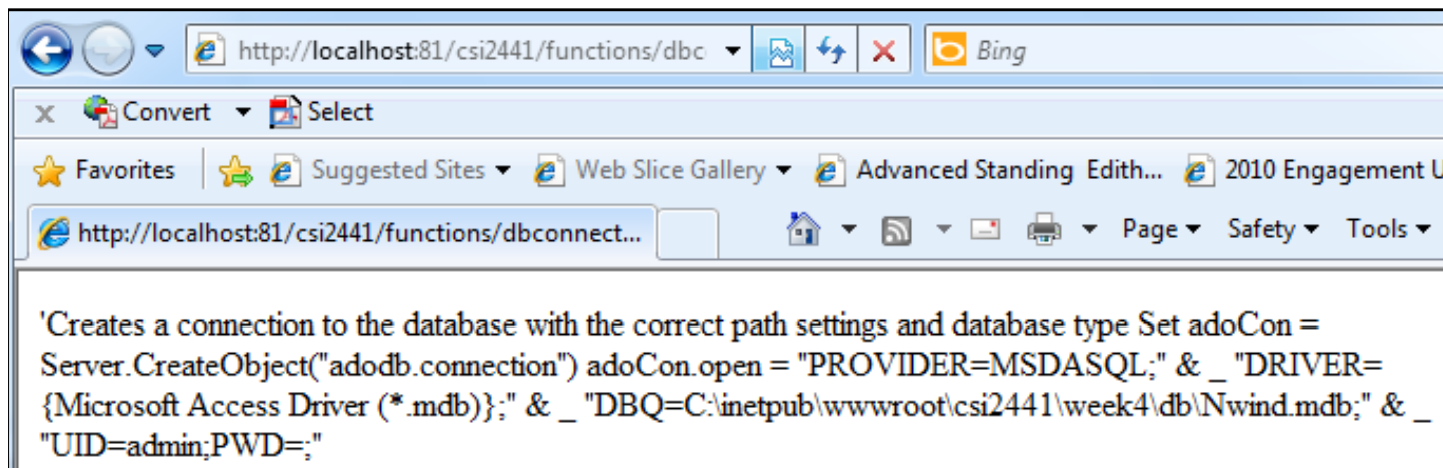
- There are some issues with storing files separately, especially in web applications
 - You must ensure the files do not sit in the website proper, so that external users can access them directly
 - This can be done by configuring a virtual folder in the web server which sits outside of the web server root
 - Also, the extension should be in the processing language, otherwise if clicked, it will display the text on screen
 - For example, if your environment is .asp, then any external function files should have .asp as the extension
 - This way, if accessed and clicked by users, they will be processed, not displayed
- The following example illustrates the issue with putting function files in publically accessible locations



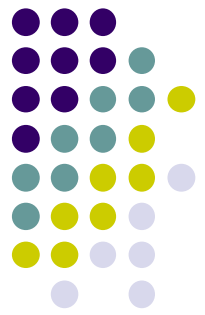
This is my database connection function file sitting in a publically accessible web folder



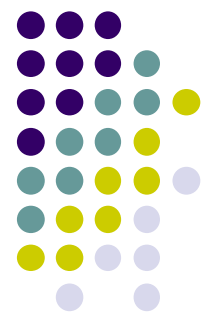
When clicked, the server tries to process it, but as it has no handler for .inc files, it then spits it out as text



Now a potential hacker knows your db location and passwords

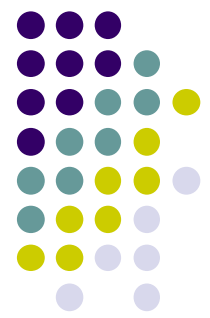


Application Features




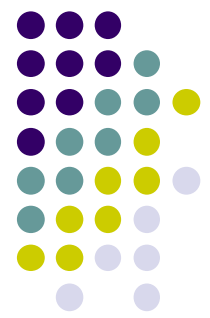
Core Features of Any Application

- While there are obviously far too many types of applications and possible features to cover here, the following slides cover some core basics
- These are more guidelines than rules but can lead to a better quality outcome
- Some seem obvious, others can rear their head only once development has commenced



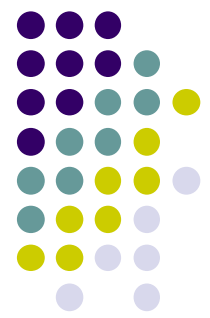
Navigation

- The role of navigation is to make the application EASIER to use, not HARDER
- Navigation style and method should be consistent – ie, do you have menus across the top, down the side, or at the bottom (not all three)
- Ensure it is the same on ALL pages
- Every page should have a link to the homepage
- Users should never have to resort to using the  browser arrows to navigate between pages
- Where possible, use 'breadcrumb' menus to show users where they are in a structure [ECU Home](#) > [Research](#) > [News](#)
- Ideally, users should be no more than 3 clicks away from homepage to desired site content



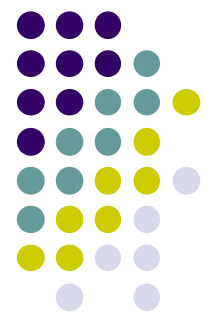
Interface

- As with navigation, it should be consistent across all pages
- Use centralised CSS style sheets to format all pages
- Keep font, colour and sizing variation to a minimum
- Do not include annoying background sounds or mouse-altering gimmicks
- Ensure images and other non-text content is appropriately sized (so it does not take forever to load)
- Try and avoid functionality that requires non-standard plug-ins to be installed
- Try and design for minimal screen resolutions – ie assume something along the lines of 1024 x 768 – users do not mind scrolling up and down, but do not like having to scroll left to right



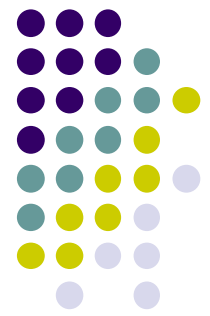
Browser Compatibility

- At a minimum your web application should run with IE, Mozilla and Safari (but the more browsers the better)
- However, the more specialist your client-side interface is, the less likely it is to work across all browsers and platforms (Active X is a classic example)
- Backwards compatibility can be a real issue – do you test against the current crop of browsers, or backtest for the last few versions?
- Regardless, this is a decision that needs to be made in conjunction with the clients, and not just by the developers to make things easier
- Organisations across the world are littered with web apps that ‘only work on this browser, this version and on this OS’ – which kind of defeats the purpose of web apps in the first place



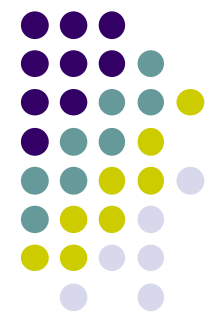
User Management

- User management can be a very time consuming element of web application coding
- There are a number of issues that arise, including;
 - Adding, editing and removing users
 - Users with different levels of access
 - Users editing their own data
 - Do you 'delete' users or merely disable their access
 - Login limits
 - Lost passwords



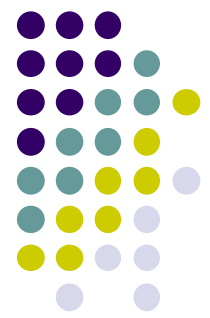
User Management

- Most systems allow users to change their own password
 - Do you enforce strong passwords / minimum length requirements?
 - Do you prevent them using previous passwords?
- When logging in, do you enforce a try limit? If so, is the lockout based on time (ie 12 hours) or does someone actually have to re-enable the account (such as an admin)
- Do you give users a password reminder question (such as Name of 1st Dog) to retrieve forgotten passwords – and if so, do you show the password on-screen, or generate a temp password and make them log in and change it?
- When creating a new user account or allowing a user to change their password, always include the password field twice, to confirm they typed it correctly



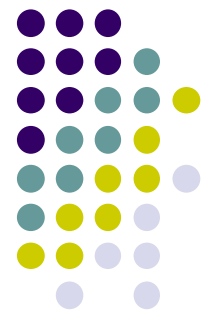
Logging

- In any application where users log-in to add, edit or delete data, a record should be kept, including
 - When users logged-in and out
 - Ip address
 - When data was added, edited or deleted
- There is no need to capture ALL data, just enough to be useful later
- Try not to capture record id's, as these may later be deleted which makes reading logs more difficult
- A simple log may be
 - "USER jbrown LOGGED-IN AT TIME 12:0:05-22-8-2010 FROM 203.123.23.99"
 - USER jbrown ADDED ITEM BrandCo Plasma TV AT TIME 12:0:09-22-8-2010 FROM 203.123.23.99"



Logging

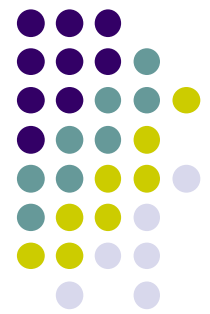
- Logs provide an audit trail and accountability
- They can also indicate patterns of unusual usage, such as
 - Multiple failed log-in attempts
 - Users logging in at unusual times
 - Patterns of incorrect usage
 - Usage patterns for load balancing (ie high and low periods)
- You can also record other details such as what browser people are using to connect to the site
- More advanced logging includes click-stream analysis whereby you can track a user from the time they enter a site until they leave
 - Especially useful for commercial sites to analyse patron browsing habits



Validation and Error Messaging

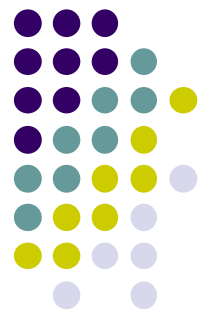
- On any form you present to a user, make it clear what is a required field
- For some fields, provide a simple example of acceptable input – for example you might have ‘dd/mm/yyyy’ next to the date field
- When the form is validated (either on the client or the server) display ALL errors in a list, not one at a time, otherwise users have to back and forth to the point where they give up
- For large forms it may be necessary to use sessions to record all form fields, so that if the user has to go ‘Back’ and fill in the form again, all the fields can be automatically re-populated

A diagram of a form with six input fields. The first two fields have a red asterisk at the end, indicating they are required. The remaining four fields are empty.



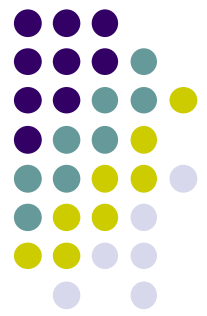
Accessibility

- People with sight impairment should be able to use the web application as well
- The World Wide Web Consortium (W3C) has guidelines of making sites usable, including the Web Content Accessibility Guidelines (WCAG) 1 & 2
- By 2015 compliance to WCAG 2.0 for Australian Government websites will be mandatory
- Depending upon the application, site owners can face discrimination claims if their site is not 'reasonably' accessible



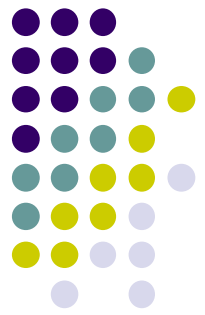
Summary

- Three steps to designing a good program:
 - Understand the output that is required
 - Ensure you have the necessary input data
 - Plan the mainline logic
- Housekeeping tasks done at the beginning of the program: declaring variables, opening files, printing headings
- Main loop is controlled by EOF decision
- Each data record passes through the main loop once



Summary (continued)

- End-of-job steps done at end of the program: printing summaries, closing files
- Good design becomes more critical as programs get larger
- Program components can be stored in separate files
- Select meaningful, pronounceable names
- Avoid confusing line breaks, use temporary variables, and use constants where appropriate



Readings

- Farrell, J. (2006). Programming Logic and Design Comprehensive, 4th Ed. Thomson : Boston. Chapter 4