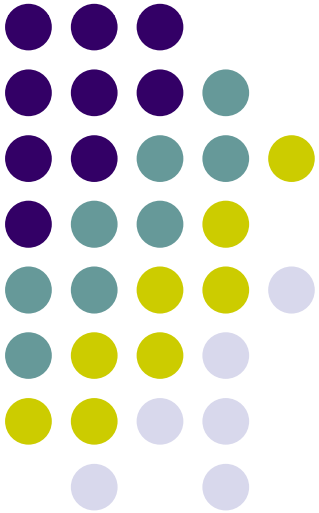
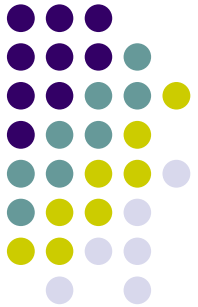
CSI2441: Applications Development

Lecture 3

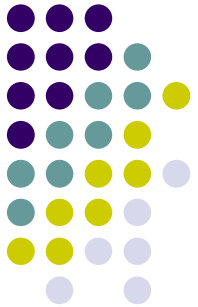
Modularization, Hierarchies and Documentation





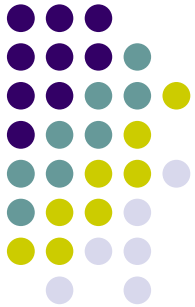
Objectives

- Describe the advantages of modularization
- Modularize a program
- Understand how a module can call another module
- Explain how to declare variables
- Create hierarchy charts



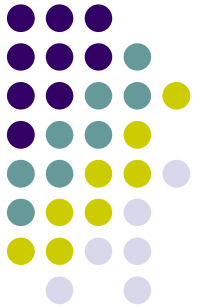
Objectives (continued)

- Understand documentation
- Design output
- Interpret file descriptions
- Understand the attributes of complete documentation



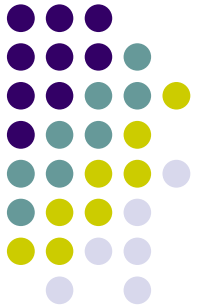
Modules, Subroutines, Procedures, Functions, or Methods

- **Module**:
  - Unit of code that performs one small task
  - Called a subroutine, procedure, function, or method
- **Modularization**: breaking a large program into modules
- Should be considered non-optional



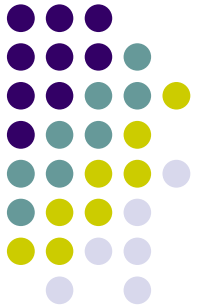
Modules, Subroutines, Procedures, Functions, or Methods (continued)

- Advantages of modularization:
  - Provides abstraction
  - Allows multiple programmers to work simultaneously
  - Allows code reuse
  - Makes identifying structures easier



Modularization Provides Abstraction

- **Abstraction**:
  - Focusing on important properties while ignoring non-essential details
  - Avoids the low-level details and uses a high-level approach
  - Makes complex tasks look simple



Modularization Provides Abstraction (continued)

- A To-Do list

with abstraction:

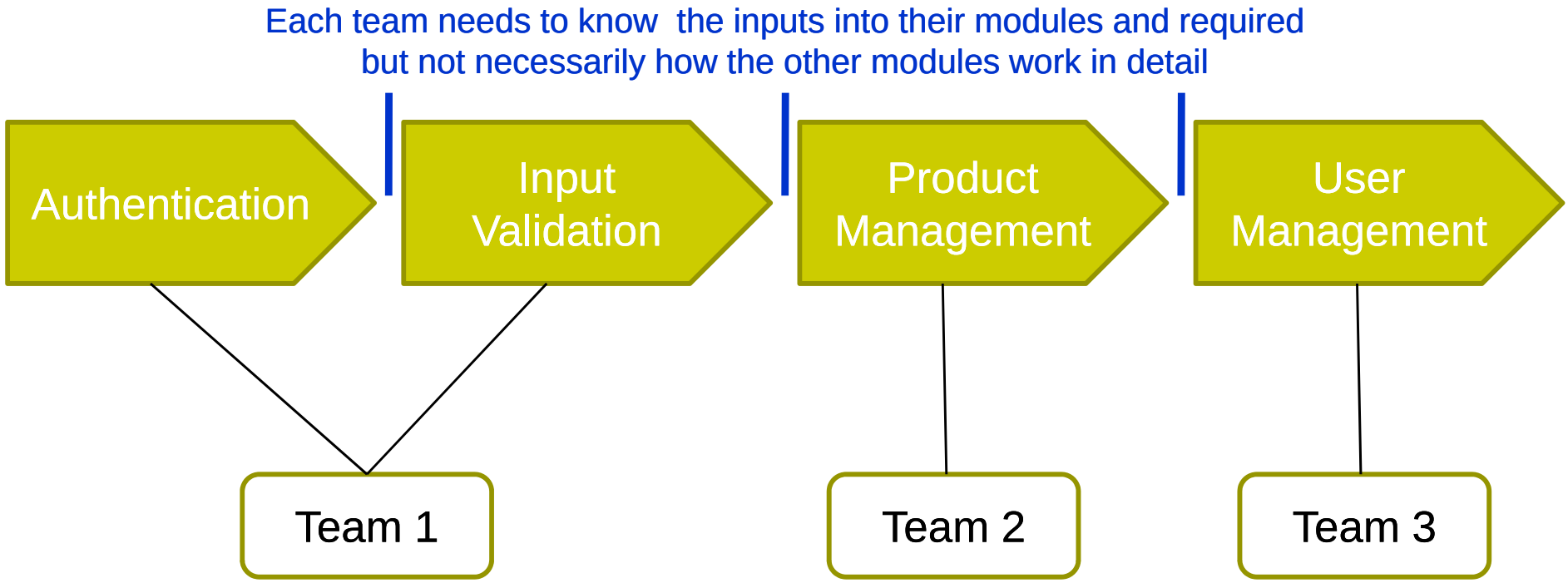
```
Do laundry
Call Aunt Nan
Start term paper
```

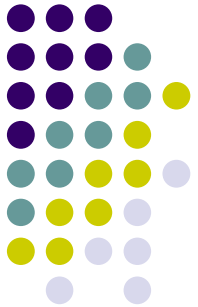
without abstraction:

```
Pick up laundry basket
Put laundry basket in car
Drive to laundromat
Get out of car with basket
Walk into laundromat
Set basket down
Find quarters for washing machine
. . . and so on.
```

Modularization Allows Multiple Programmers to Work on a Problem

- Large programming projects can be divided into modules
- Modules can be written by different programmers
- Development time is significantly reduced

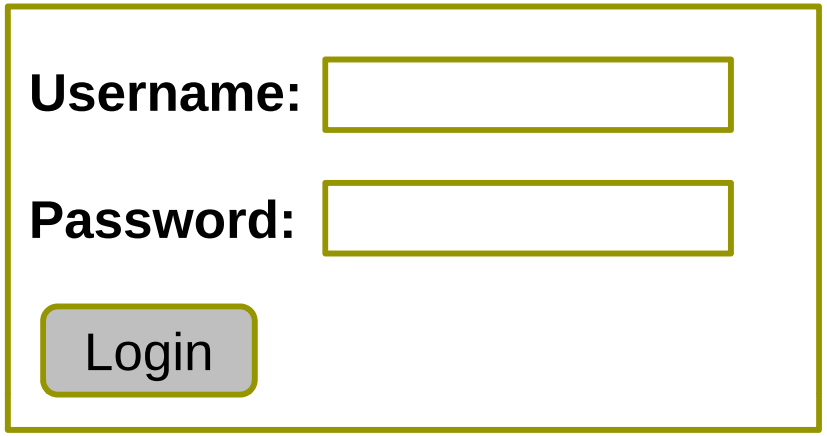


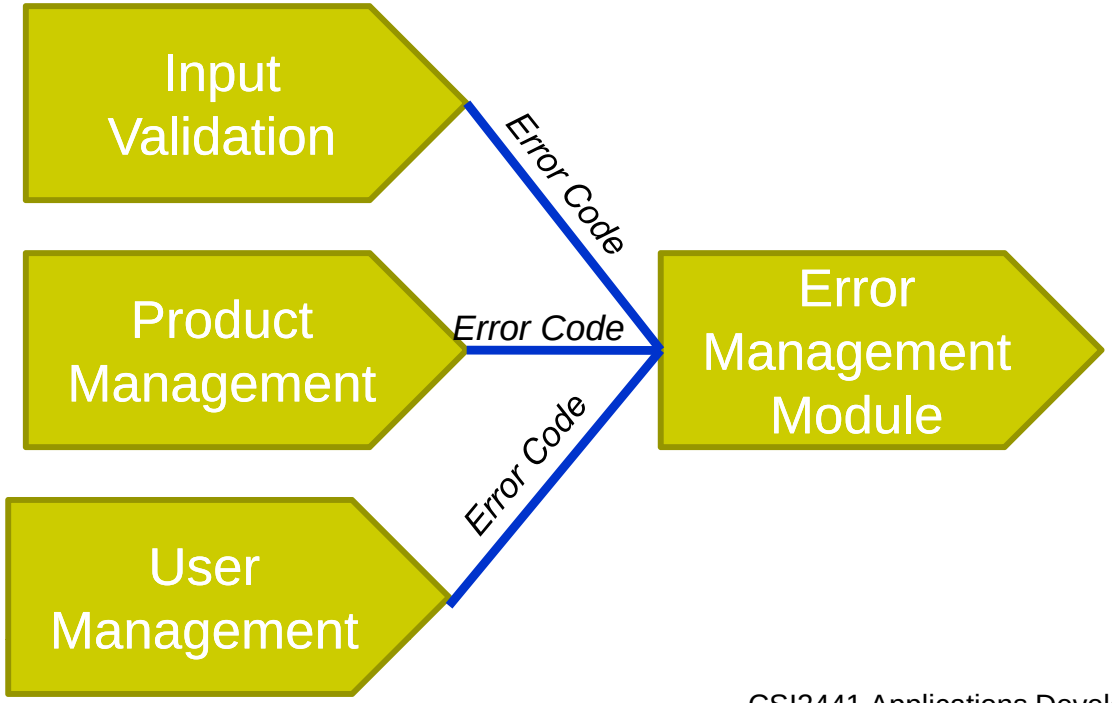


Modularization Allows You to Reuse Your Work

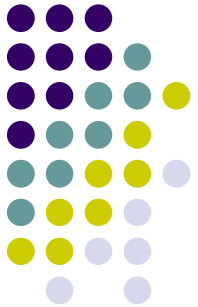
- **Reusability:** the ability to use modules in a variety of applications
  - i.e error trapping, logging, user authentication
- **Reliability**: assurance that a module has been tested and proven to function correctly
  - If the module has been written as a standalone entity, then if it is linked to other code and errors occur, the linking or other code is the first place to look for problems
- **Efficiency:** makes it much easier to locate code elements for code maintenance. Also, one instance of a function – thus one change only to change an application-wide function
  - again, if an error trapping and management module is used, there is one for the entire application, not each section of the application

Reusability, Reliability, Efficiency





Errors from all other modules would be sent to this module, which would deal with them appropriately. Any additions to or changes in error management processes occur only in this module.

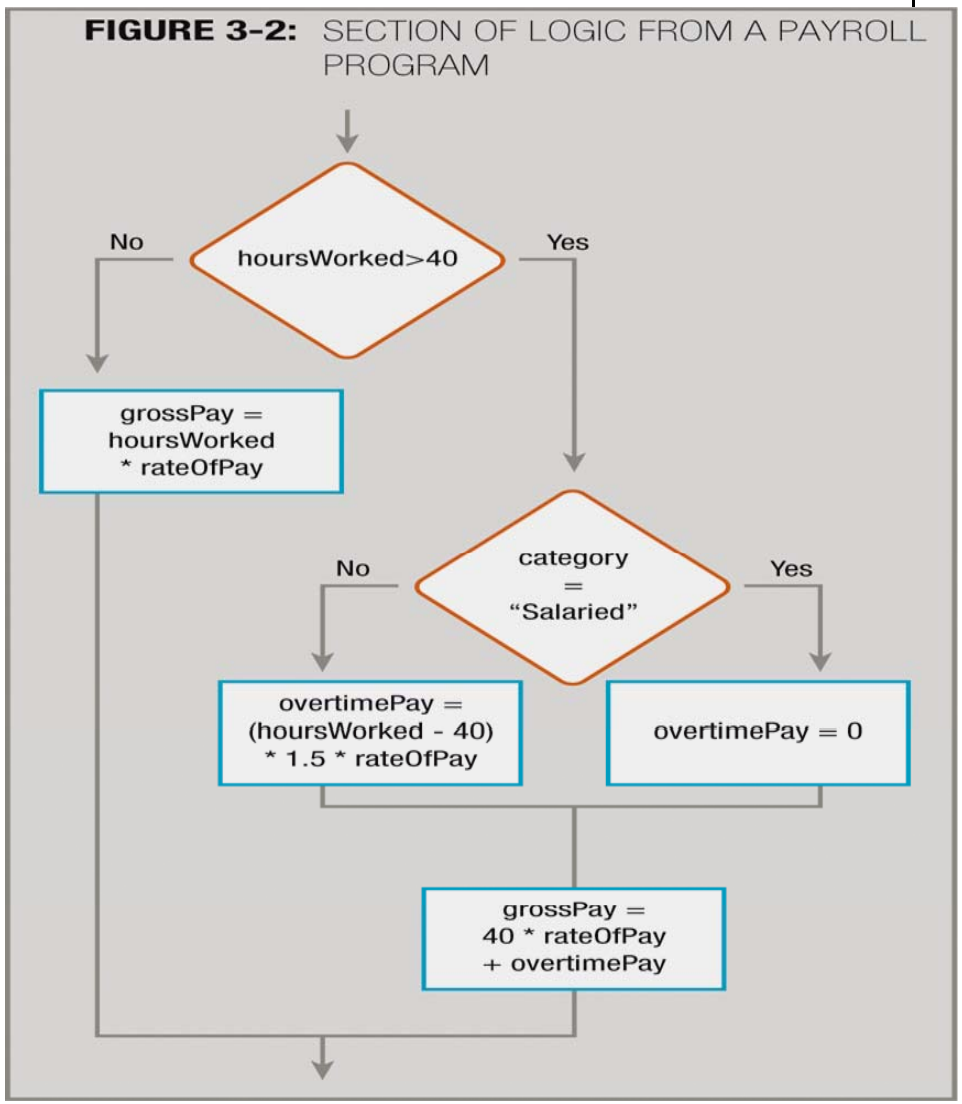


Modularization Makes It Easier to Identify Structures

- Combining several tasks into modules may make it easier for beginning programmers to:
  - Determine if a program is structured
  - Identify structures in a program
- Experienced programmers modularize for abstraction, ease of dividing the work, and reusability

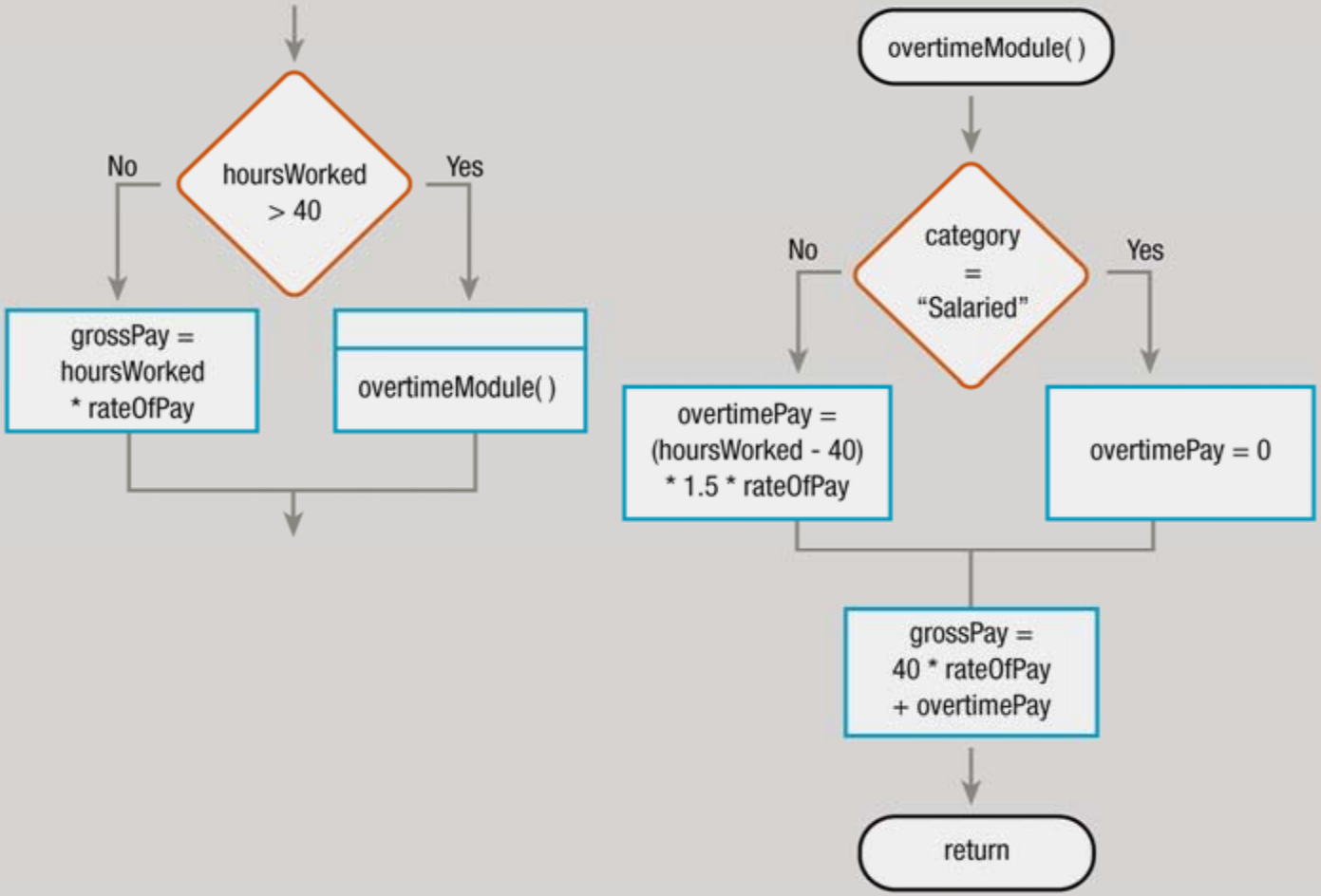
Modularization Makes It Easier to Identify Structures (continued)

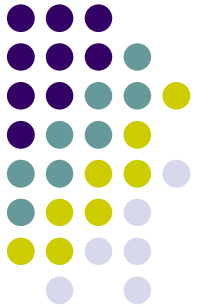
- Minimal structure



Modularization Makes It Easier to Identify Structures (continued)

FIGURE 3-3: MODULARIZED LOGIC FROM A PAYROLL PROGRAM





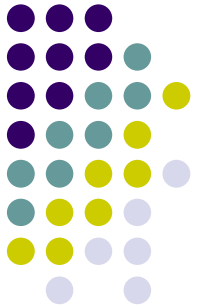
Modularizing a Program

- Most programs contain a **main module**
  - Contains the **mainline logic**
  - Accesses other modules or subroutines as required
  - In most structures, subroutines are called, do their job, and return the app back to the mainline logic
- Rules for module names used here:
  - Must be one word
  - Should be meaningful
  - Are followed by a set of parentheses

Modularizing a Program (continued)

TABLE 3-1: VALID AND INVALID MODULE NAMES FOR A MODULE THAT CALCULATES AN EMPLOYEE'S GROSS PAY

Suggested module names for a module that calculates an employee's gross pay	Comments
`calculateGrossPay()`	Good
`calculateGross()`	Good—most people would interpret "Gross" to be short for "Gross pay"
`calGrPy()`	Legal, but cryptic
`calculateGrossPayForOneEmployee()`	Legal, but awkward
`calculate gross()`	Not legal—embedded space
`calculategrosspay()`	Legal, but hard to read without camel casing

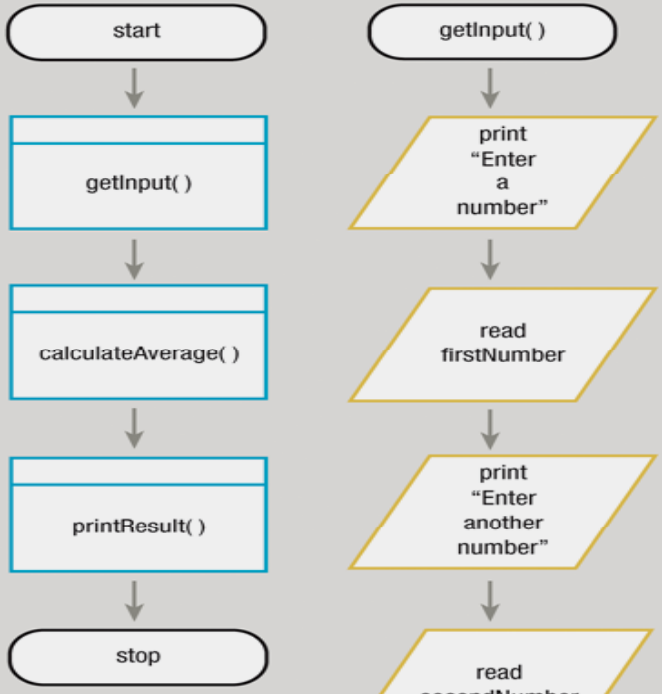


Modularizing a Program (continued)

- **Calling program** (or **calling module**): one that uses another module
- Flowchart symbol for calling a module: a rectangle with bar across the top
- Flowchart for the module contains:
  - Module name in the start symbol
  - **`exit`** or **`return`** in the stop symbol
- When a module is called, logic transfers to the module
- When module ends, logic transfers back to the caller

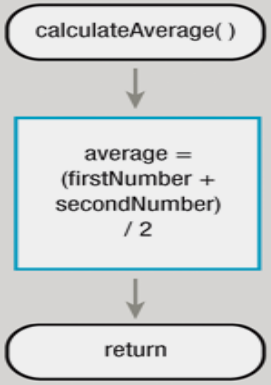
Modularizing a Program (continued)

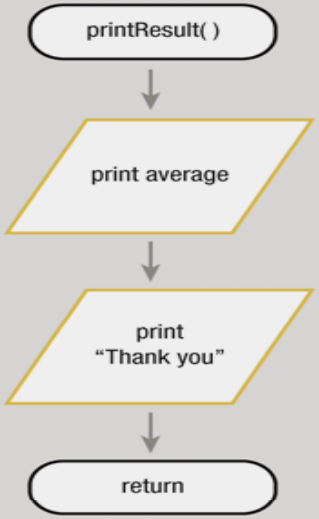
FIGURE 3-4: FLOWCHART AND PSEUDOCODE FOR AVERAGING PROGRAM WITH MODULES




```
start
      perform getInput()
      perform calculateAverage()
      perform printResult()
stop

getInput()
      print "Enter a number"
      read firstNumber
      print "Enter another number"
      read secondNumber
return

calculateAverage()
      average = (firstNumber + secondNumber) / 2
return

printResult()
      print average
      print "Thank you"
return
```

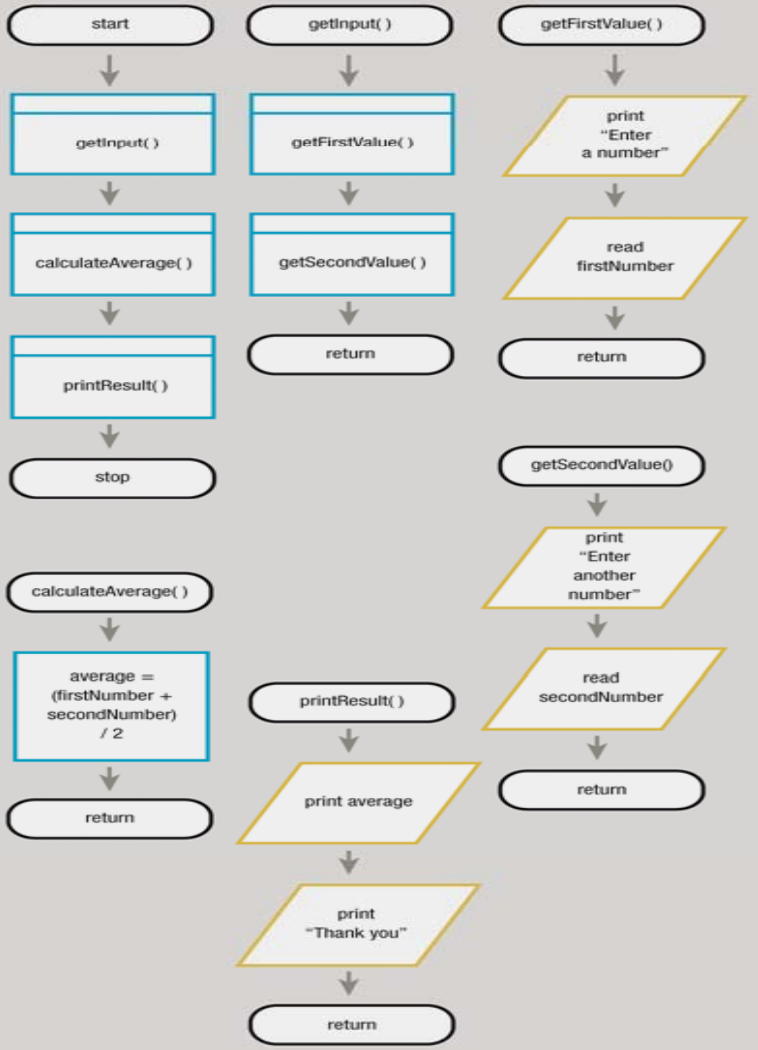
Mainline Logic



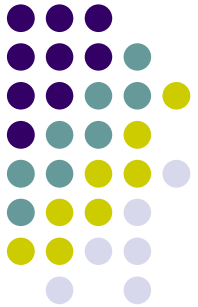


Modules Calling Other Modules

FIGURE 3-5: FLOWCHART AND PSEUDOCODE FOR AVERAGING PROGRAM WITH SUBMODULES

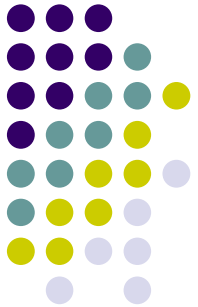



```
start
   perform getInput()
   perform calculateAverage()
   perform printResult()
stop

getInput()
   perform getFirstValue()
   perform getSecondValue()
return

getFirstValue()
   print "Enter a number"
   read firstNumber
return

getSecondValue()
   print "Enter another number"
   read secondNumber
return

calculateAverage()
   average = (firstNumber + secondNumber) / 2
return

printResult()
   print average
   print "Thank you"
return
```



Modules Calling Other Modules (continued)

- Knowing when to break a module into its own subroutines or submodules is an art
- Best practice: place together statements that contribute to one specific task
- **Functional cohesion**: extent to which the statements contribute to the same task
- Over-modularised applications can be as difficult to manage as under-modularised ones

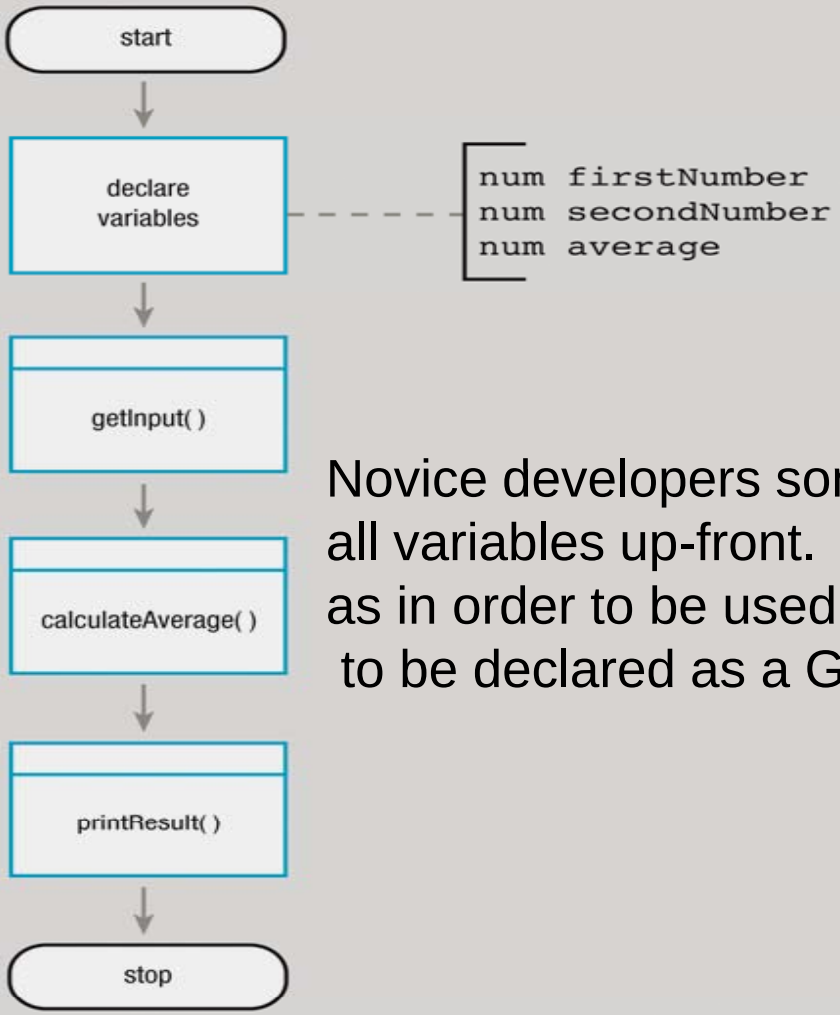


Variables and Modularisation

- **Local variables:** declared within a module and only work within that module
- **Global variables:** declared at the beginning of the program, and used in all modules
- **Data Dictionary**: list of variables used in a program, with their type, size, and description and typically which modules use them

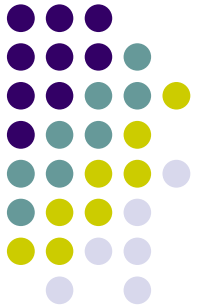
Declaring Variables (continued)

FIGURE 3-6: FLOWCHART AND PSEUDOCODE FOR MAINLINE LOGIC FOR AVERAGING PROGRAM SHOWING DECLARED VARIABLES




```
start
   declare variables
        num firstNumber
        num secondNumber
        num average
   perform getInput()
   perform calculateAverage()
   perform printResult()
stop
```

Novice developers sometimes create modules to define all variables up-front. However this is not a valid practice as in order to be used, each of the variables would have to be declared as a Global variable.

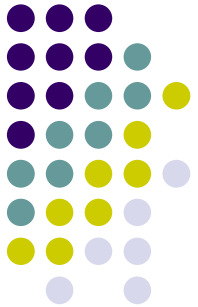


Passing Variables as Parameters

- In most instances of calling a module, you need to send it data to work with

```
varWorkingWeek = 40
varWorkedHours = 50

If varWorkedHours > varWorkingWeek then
        call OvertimeModule(varWorkedHours,varWorkingWeek)
End if

Module OvertimeModule(varWorkedHours, varWorkingWeek)
        varTotalOvertime = varWorkedHours – varWorkingWeek
        return varTotalOvertime
End

Print “You worked a total of “ . varTotalOvertime . “ hours of overtime”
```

- In this instance the mainline logic has called the overtime module, sent it some data to work with and received back a computed value



Calling modules without parameters

- If your module does not need any input, then you can call it as is

```
Call LogoutModule()

Module LogoutModule()
        DestroySession
        Print “You have been logged out”
        exit
End
```

- Another option is where you call a module with parameters, but do not return to the mainline logic

```
Call ErrorModule(varErrorType=“1”)

Module ErrorModule(varErrorType)
        Print “The following error has occurred “ . varErrorType
        Print “ and the program is exiting.”
        exit
End
```

Creating Hierarchy Charts

- **Hierarchy chart**:
  - Illustrates modules' relationships
  - Tells which routines call which other routines
  - Does not tell when or why the modules are called

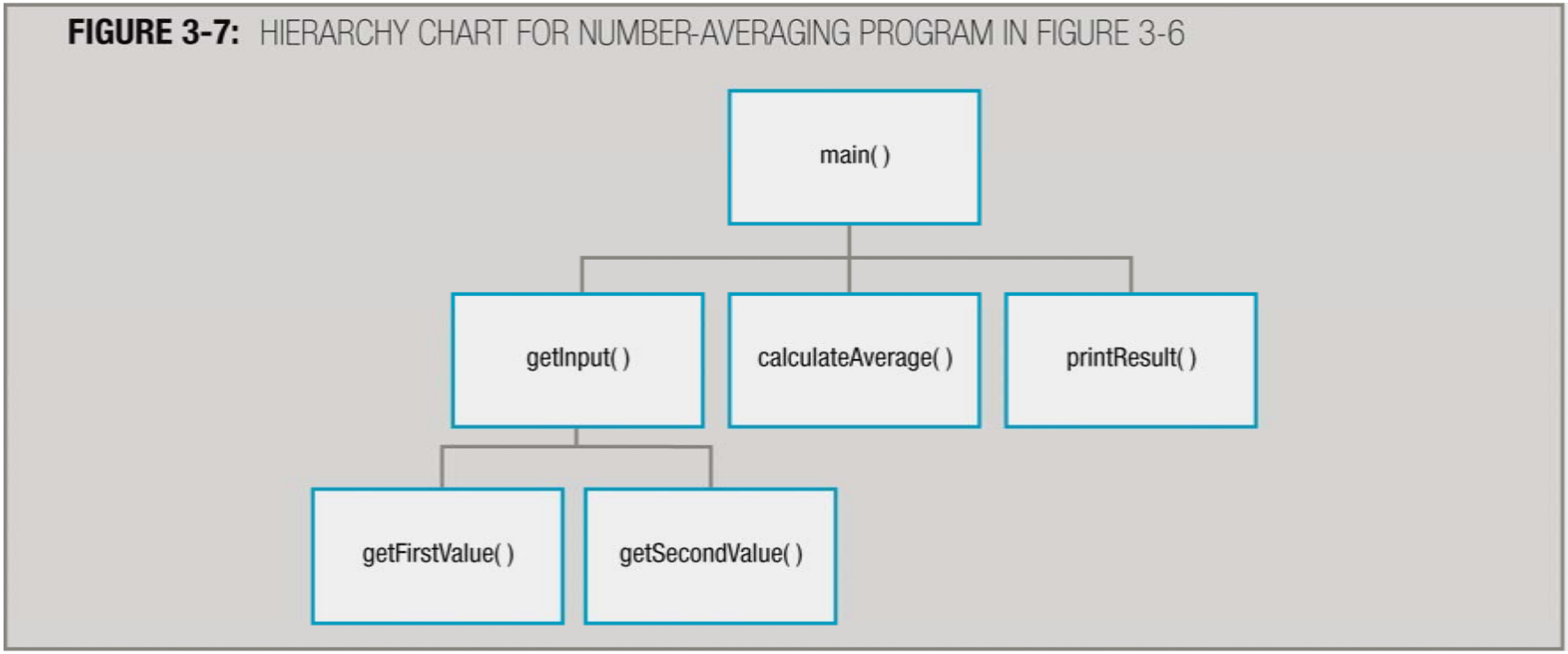
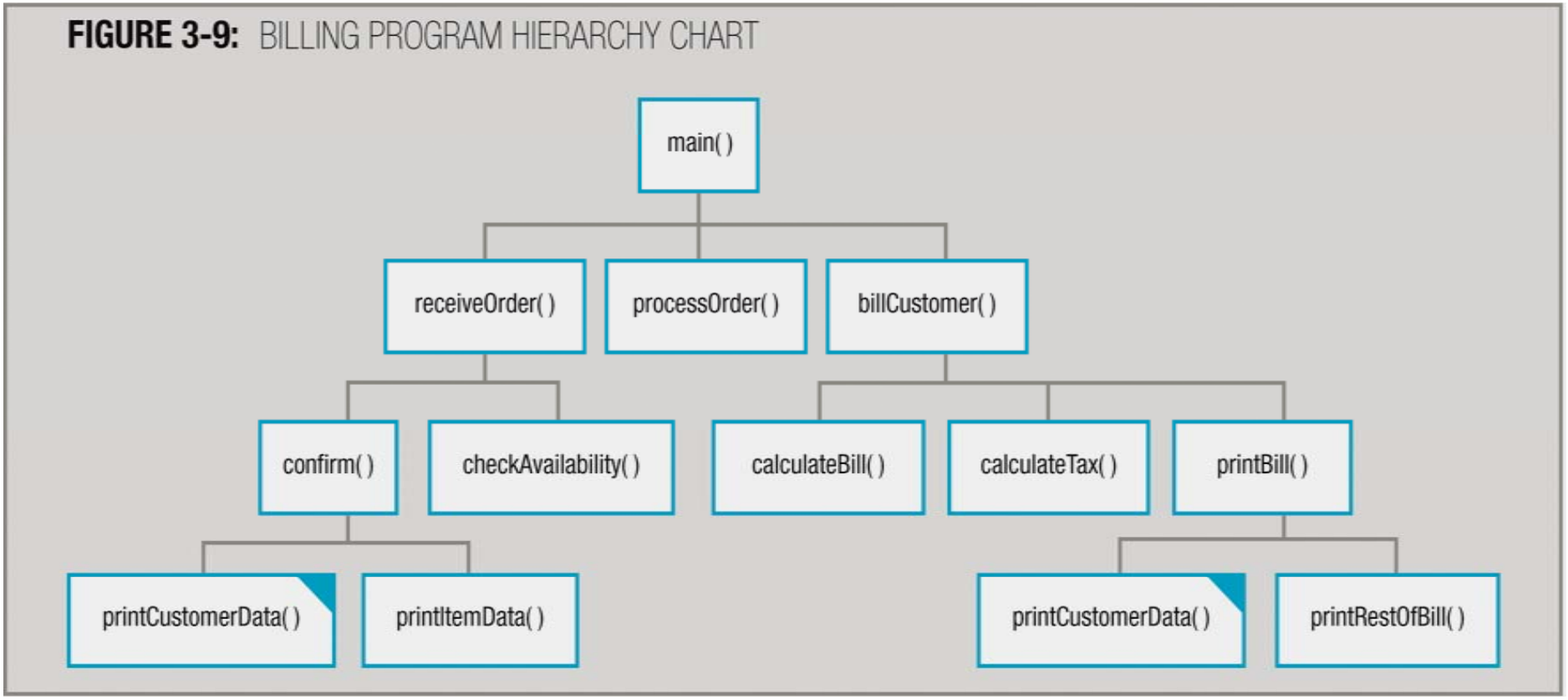
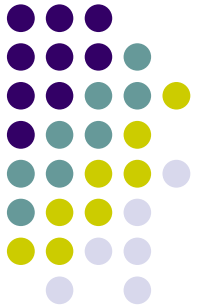


FIGURE 3-7: HIERARCHY CHART FOR NUMBER-AVERAGING PROGRAM IN FIGURE 3-6

Creating Hierarchy Charts (continued)

- Blackened corner indicates a module that is used more than once





Understanding Documentation

- **Documentation**:
  - All supporting material that goes with a program
  - Two major categories: for users and for programmers
  - Usually created by system analysts and/or tech writers
  - May be printed or electronic (Web or CD)
- **End users**: people who use computer programs
- **Program Documentation**:
  - Internal program documentation: comments within code
  - External program documentation: supporting paperwork written before programming begins

Code Commenting

- Comments should be short, sharp, sweet and to the point
- Do not just repeat the code that you are trying to comment
- Comments come before the code or on the same line, but not after

```
    String s = "Wikipedia"; /* Assigns the value "Wikipedia" to the variable s. */
```

http://en.wikipedia.org/wiki/Comment_(computer_programming)#Necessity_of_comments

- Or

```
/*
 * Check if we are over our maximum process limit, but be sure to
 * exclude root. This is needed to make it possible for login and
 * friends to set the per-user process limit to something lower
 * than the amount of processes root is running. -- Rik
 */
if (atomic_read(&p->user->processes) >= p->rlim[RLIMIT_NPROC].rlim_cur
    && !capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE))
    goto bad_fork_free;
```

http://en.wikipedia.org/wiki/Comment_(computer_programming)#Necessity_of_comments

- In this case we also align comments to the left and include basic name details of the developer who wrote the code

Output Documentation

- Usually written first
- Represents the information needed by end users
- May be initially created by end users
- Printed reports: designed using a **print chart**
- This is how you would plan for printed outputs

FIGURE 3-10: PLANNED PRINT CHART

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
1																																																													
2											I	N	V	E	N	T	O	R	Y		R	E	P	O	R	T																																							
3																																																																		
4				I	T	E	M		N	A	M	E										P	R	I	C	E					Q	U	A	N	T	I	T	Y		I	N		S	T	O	C	K														
5																																																																		
6				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			9	9	9	.	9	9													9	9	9	9																					
7				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			9	9	9	.	9	9													9	9	9	9																					
8																																																																	
9																																																																	
10																																																																	
11																																																																	
12																																																																	
13																																																																	
14																																																																	

Output Documentation (continued)

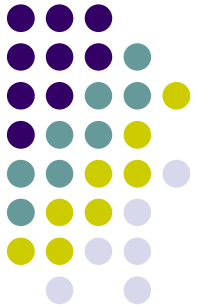
- Reports to screen are typically more flexible as you do not have to fit-to-page
- As with printed reports you will have headings, totals and other summary data
- However, you can also enhance the output by linking to more detailed reports

Inventory Report

Item Name	Price	Quantity in Stock	
2.5" HDD 160Gb	*$82.00*	*17*	*More Info*
3.5" HDD 1.5Tb	*$129.00*	*25*	*More Info*
24" TFT Display	*$289.00*	*3*	*More Info*

Item Report

Item Name	Price	Quantity in Stock	On Order	Sales YTD
2.5" HDD 160Gb	*$82.00*	*17*	*10*	*83*

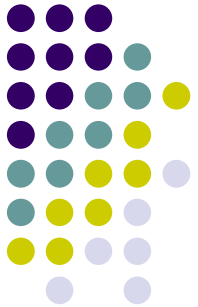


Input Documentation

- **Input documentation**: describes what input is available to produce the output
- **Database description**:
  - Describes the data stored in a file
  - Indicates fields, data types, and lengths

```
FIELD DESCRIPTION      DATA TYPE    COMMENTS
Name of item           Character    15 bytes
Price of item          Numeric      2 decimal places
Quantity in stock      Numeric      0 decimal places
```

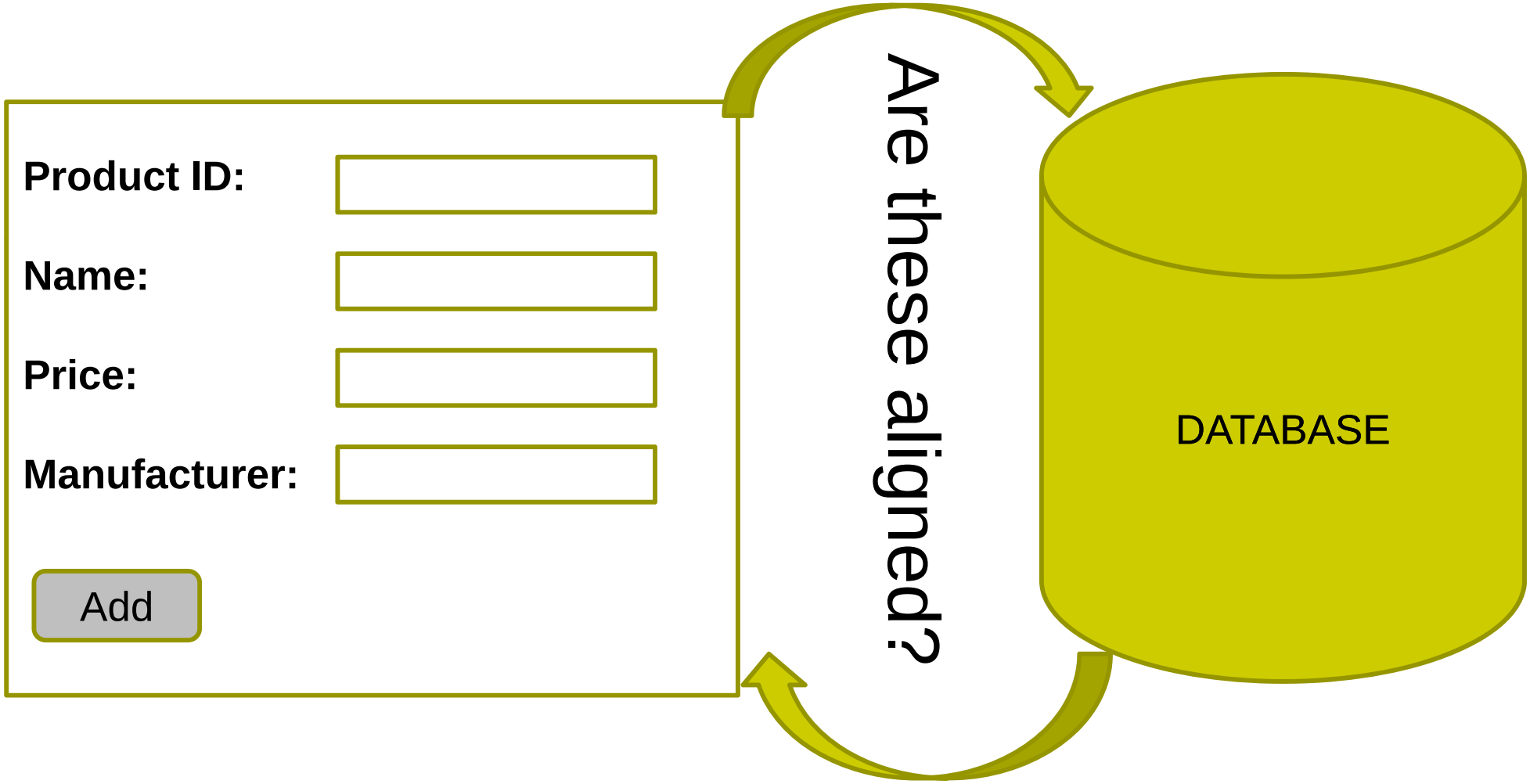
- You must always reflect the field lengths and data type requirements of your database in your web forms
- In this example, if your form allows > 15 characters to be entered for the Name of an item, you will cause a data insert error on your database

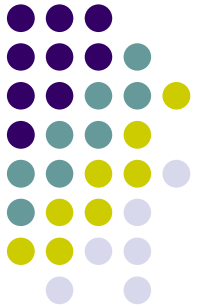


Input Documentation (continued)

- So, in your user documentation and code commenting you need to indicate for each field in a database table;
  - Does it allow nulls?
  - What is the datatype?
  - Is there a minimum and maximum length
  - Is there a set number of possible inputs (i.e M or F for Male or Female)?
  - Is the data reasonable? (i.e would you allow a Date of Birth of 1-1-1850)
  - Is the data in the correct form (i.e dd/mm/yyyy OR mm/dd/yyyy)
- Thus we are checking for the presence of data, the type of data and the reasonableness of data
- Some of these things can be checked on the client-side before the form is submitted, others need to processed within the server-side business logic of the application

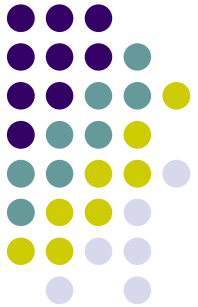
Input Documentation (continued)





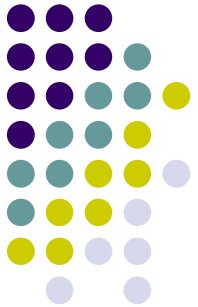
Completing the Documentation

- Program documentation may contain:
  - Output design
  - Input description
  - Flowcharts
  - Pseudocode
  - Program code listing
  - Data Dictionary
  - ERD's
- User documentation may contain
  - Manuals (one for each functional section of the app)
  - Instructional material (tested with users)
  - Operating instructions (lots of screenshots)



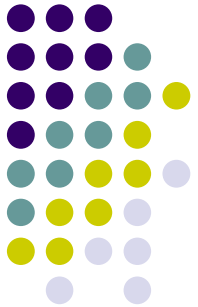
Completing the Documentation (continued)

- User documentation:
  - Written clearly in plain language
  - Usually prepared by system analysts and/or tech writers (preferably not the programmers ☺)
- User documentation usually indicates:
  - How to prepare input
  - User navigation
  - Allowable data
  - FAQ's



Summary

- Modules, subroutines, procedures, functions, methods: smaller, reasonable units of code that provide reusability
- Modules can call other modules
- Variable declarations define the name and type of the data to be stored
- Hierarchy chart illustrates modules' relationships
- Documentation: all supporting material for a program



Summary (continued)

- Output documentation: includes report designs
- File description: details data types and lengths of each field of data in the file
- User documentation: includes manuals and instructional materials for non-technical people, and operating instructions for operators and data-entry people