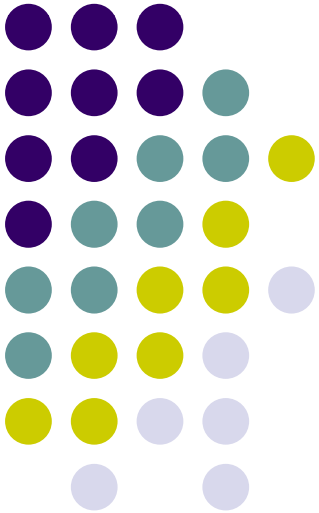
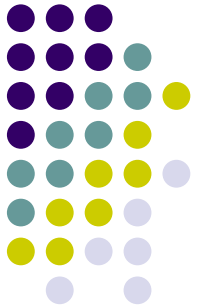
CSI2441: Applications Development

Lecture 1

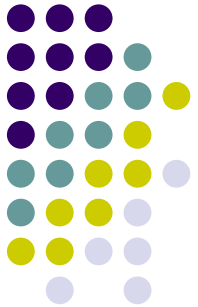
An Overview of Programming Logic and Design





Objectives

- Describe the steps involved in the programming process
- Describe the data hierarchy
- Use and name variables
- Use a sentinel, or dummy value, to end a program
- Use a connector symbol
- Assign values to variables
- Recognize the proper format of assignment statements
- Describe data types
- Understand the evolution of programming techniques

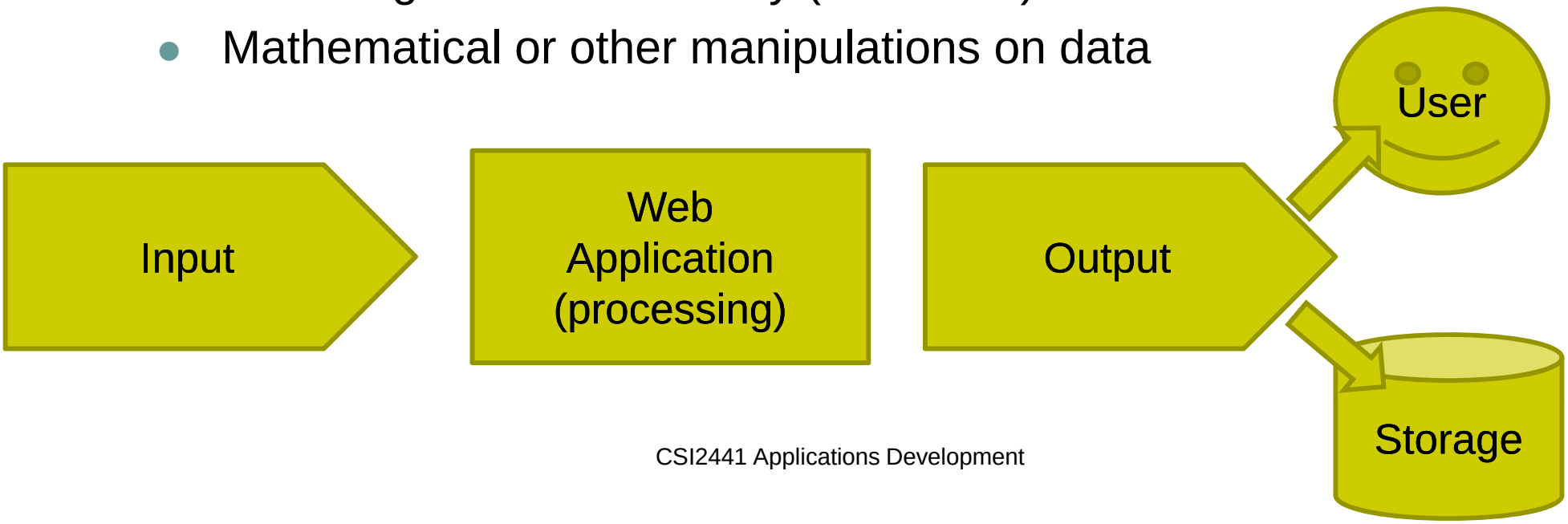


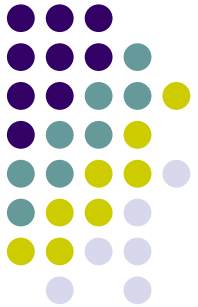
Understanding Computer Components and Operations

- Hardware and software: the two major components of any computer system
- **Hardware**: equipment, or devices
- **Software**: programs that contain instructions for the computer
- Four major operations in a computer:
  - Input
  - Processing
  - Output
  - Storage
- In this unit we are mainly concerned with Input into web applications
- Processing of that input within the application
- Output to screen of the results of that processing
- Storage and retrieval of processed data from databases

Understanding Application Components and Operations (continued)

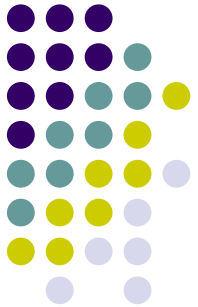
- **Input** : allow data to enter the application
  - Typically from a user, another application or a datasource (such as a database)
- **Processing**: working on the data; such as:
  - Organizing data
  - Checking data for accuracy (validation)
  - Mathematical or other manipulations on data





Understanding Application Components and Operations (continued)

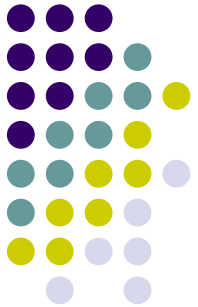
- **Output devices**: typically the web browser
- **Programming languages**: any number of web-based or specialised languages including
  - VB.Net, Java, C#, PHP, Perl…..
- **Syntax**: the rules governing word usage and punctuation in the language
- **Machine language**: a language that controls the computer's on/off circuitry
- **Compiler** or **interpreter**: software that translates programming languages to machine language
- Some web-based languages are interpreted, where an external program reads a script on-the-fly and executes as machine code on the fly
- Other languages and environments allow for compiled code (ie like executables) which allow for faster and more efficient performance



Understanding Application Components and Operations (continued)

- A program must be free of syntax errors to be run, or **executed**, on a computer
- To function properly, the **logic** must be correct
- What's wrong with this logic for making a cake?

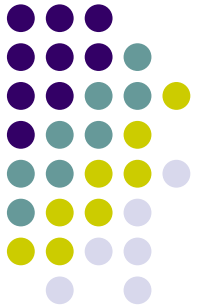
```
Stir
Add two eggs
Add a gallon of gasoline
Bake at 350 degrees for 45 minutes
Add three cups of flour
```



Understanding Computer Components and Operations (continued)

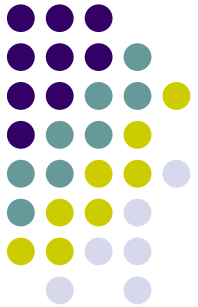
- **Logic errors**, or **semantic errors**, are more difficult to locate than syntax errors
- Most errors in large applications are due to logic errors
- Logic for multiplying a number by 2 (includes input, processing and output statements)

```
Get inputNumber.
Compute calculatedAnswer as inputNumber times 2.
Print calculatedAnswer.
```



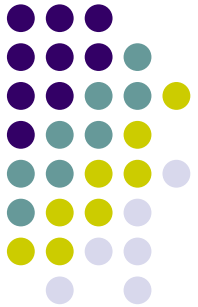
Understanding Application Components and Operations (continued)

- Business rules or business logic is how an organisation or business operates
- When an application is commissioned, the business rules for how the business works need to be translated into programmatic logic
- This is the make or break of any large application and where most errors occur
- If the original system analysis is not done correctly, business rules and processes can be missed or misinterpreted
- Fixing or adding business logic, especially where large numbers of dependencies exist, can be hugely problematic



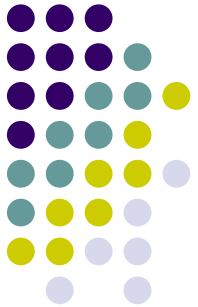
Understanding Application Components and Operations (continued)

- For most applications we use two storage categories: internal and external
- **Internal storage**:
  - Variables
  - Constants (non changing variables)
  - Sessions (storing data between web pages)
- **External storage**:
  - Basic text files
  - XML files
  - Relational Database Management System (RDBMS)



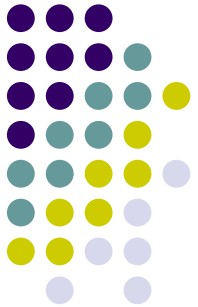
Understanding the Programming Process

- Six programming phases:
  1. Understand the problem
  2. Plan the logic
  3. Code the program
  4. Use software to translate the program to machine language
  5. Test the program
  6. Deploy the program into production



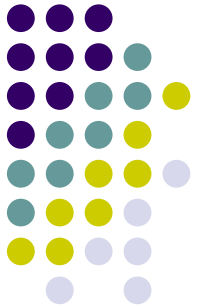
Understanding the Programming Process (continued)

- Understanding the problem:
  - May be the most difficult phase
  - Users may not be able to articulate their needs well (ie represent their company's business logic)
  - User needs may be changing frequently
  - Programmers may have to learn the user's functional job tasks
  - Failure to understand the problem is the major cause of most project failures
  - Image a large international company or large university – understanding and coding all processes, relationships between processes and dependencies between processes (ie what needs to happen and in what order)



Understanding the Programming Process (continued)

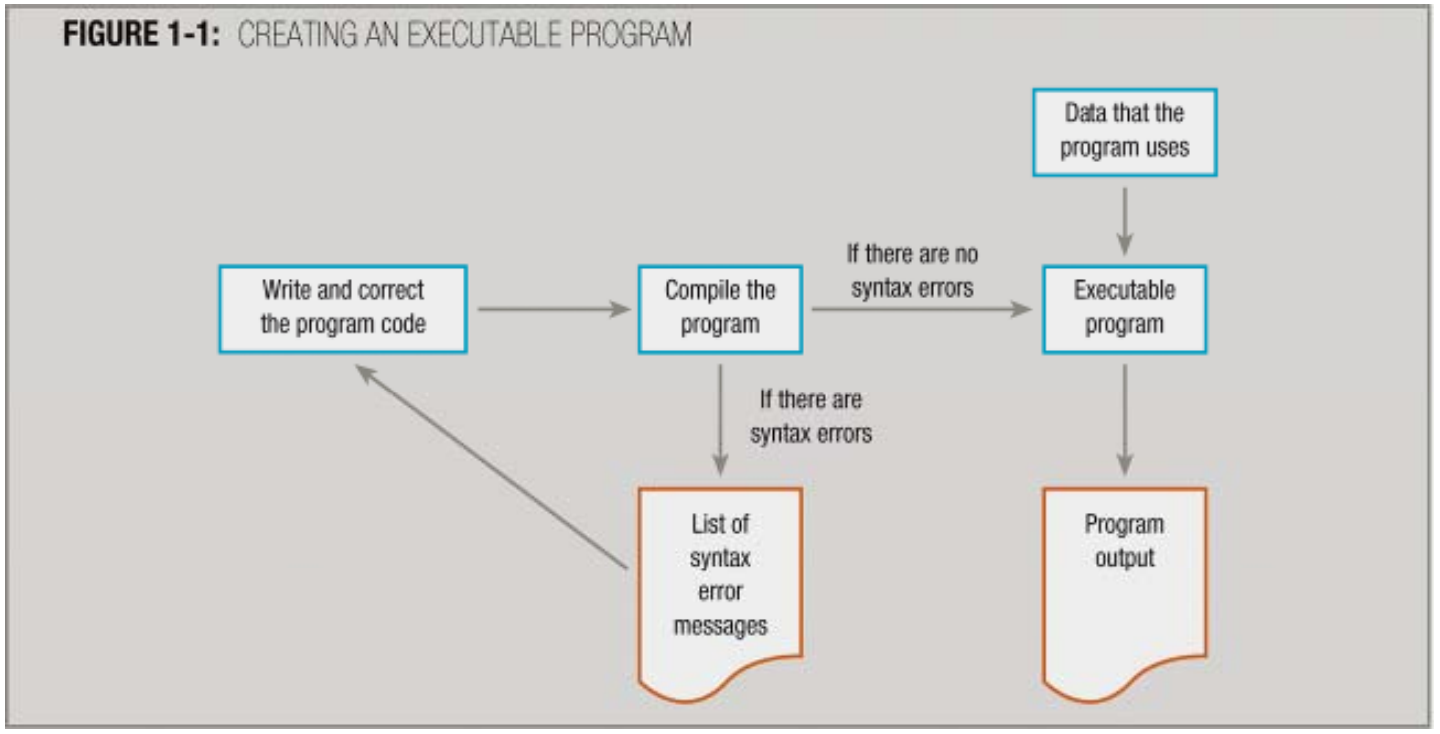
- Planning the logic:
  - Plan the steps that the program will take
  - Use tools such as flowcharts and pseudocode
  - **Flowchart**: a pictorial representation of the logic steps
  - **Pseudocode**: English-like representation of the logic
  - Walk through the logic before coding by **desk-checking** the logic
  - Do screen mockups
  - Perhaps even do form-driven analysis
    - This is where you take the forms or documents a business uses and then code requirements and interfaces based on them

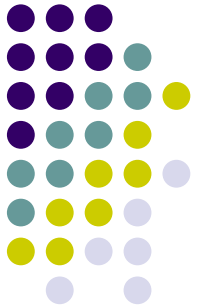


Understanding the Programming Process (continued)

- Coding the program:
  - Select the programming language
  - Write the instructions
- Using software to translate the program into machine language:
  - Programmers write instructions in English-like high-level languages
  - Compilers or interpreters change the programs into low-level machine language that can be executed
  - In some environments you will have a choice, and in others you will not
  - Syntax errors are identified by the compiler or interpreter
  - Logic errors are not, and must be identified through extensive testing

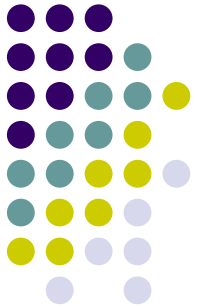
Understanding the Programming Process (continued)





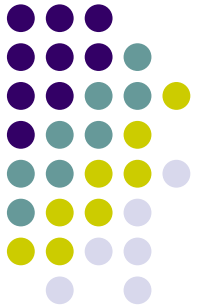
Understanding the Programming Process (continued)

- Testing the program:
  - Execute it with sample data and check results
  - Identify logic errors and correct them
  - Choose test data carefully to exercise all branches of the logic
  - Always ensure end-users get to play with the system before deployment – no one will find errors quicker ☺
- Putting the program into production
  - Do this after testing is complete and all known errors have been corrected
  - May require coordination with other related activities or software
  - Depending upon the contract, you might also be required to develop and implement a training program



Understanding the Data Hierarchy

- **Data hierarchy**: ordering of data types by size
  - **Character**: single symbol (letter, number, special symbol)
    - “A”, “7”, “$”
  - **Field**: group of characters forming a single data item
    - “Smith”
  - **Record**: a group of related fields
    - Customer record containing name and address fields
  - **Table**: a group of related records
    - Customer file, containing all customer records
  - **Database**: collection of related tables, that serve the information needs of the organization



Using Flowchart Symbols and Pseudocode Statements

- **Flowchart**: pictorial representation of the logic
  - This helps development teams understand the logic and flow of a program
  - Useful in multi-team development
  - Can also help illustrate to clients what is going on
- **Pseudocode**: English-like representation of the logic
  - Example:

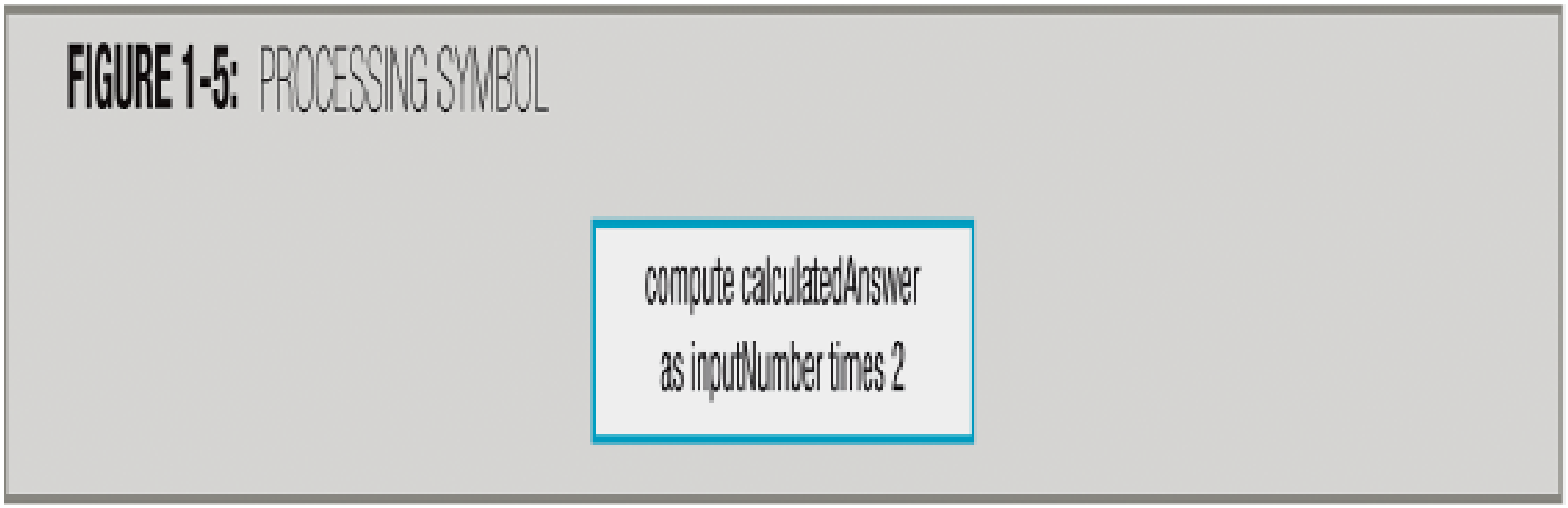
```
start
  get inputNumber
  compute calculatedAnswer as inputNumber times 2
  print calculatedAnswer
stop
```

Using Flowchart Symbols and Pseudocode Statements (continued)

- Flowchart **input symbol**:

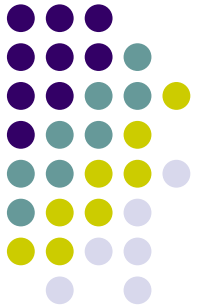
Using Flowchart Symbols and Pseudocode Statements (continued)

- Flowchart **processing symbol**



Using Flowchart Symbols and Pseudocode Statements (continued)

- Flowchart **output symbol**:

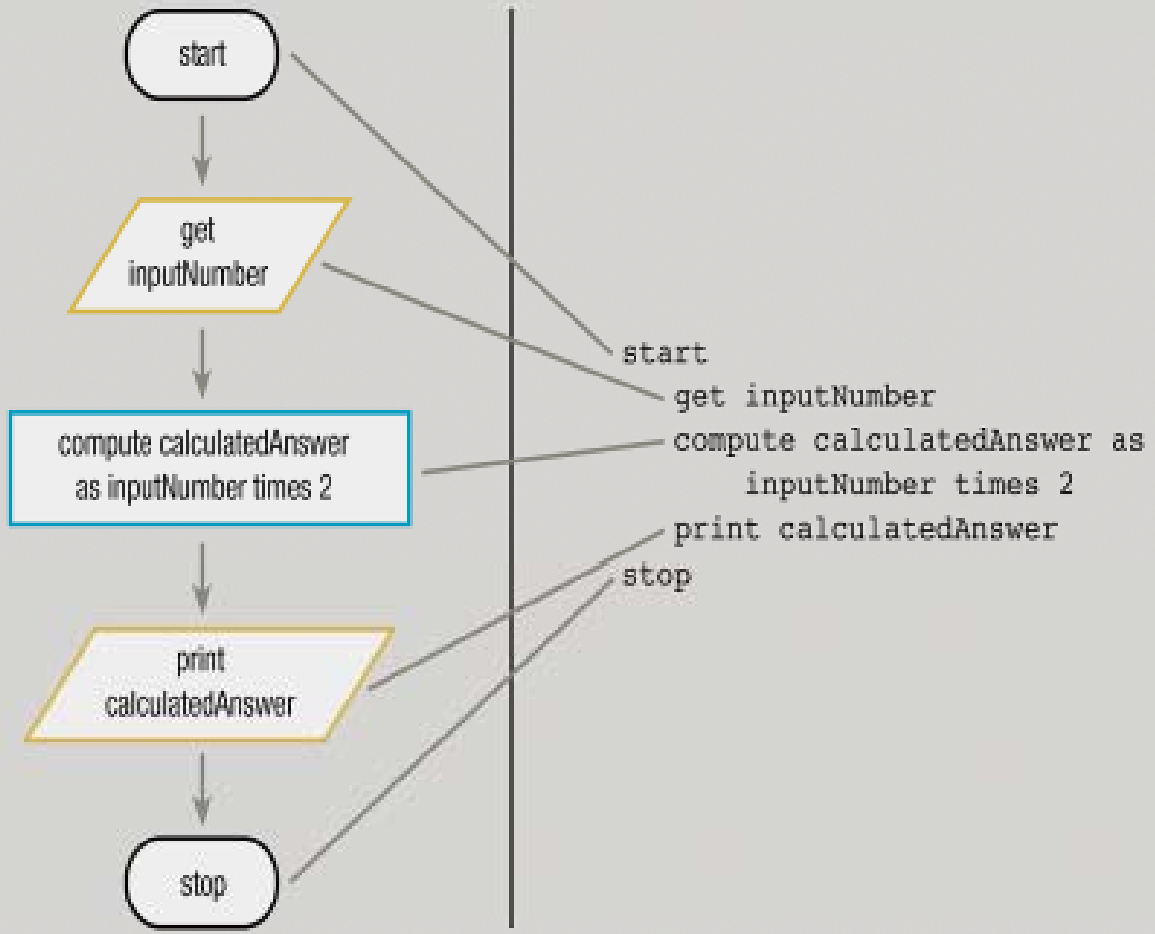


Using Flowchart Symbols and Pseudocode Statements (continued)

- **Flowlines**:
  - Connect the steps
  - Show the sequence of statements
  - Have arrows to show the direction
- **Terminal symbol** (start/stop symbol):
  - Shows the start and end points of the statements
  - Lozenge shape

Using Flowchart Symbols and Pseudocode Statements (continued)

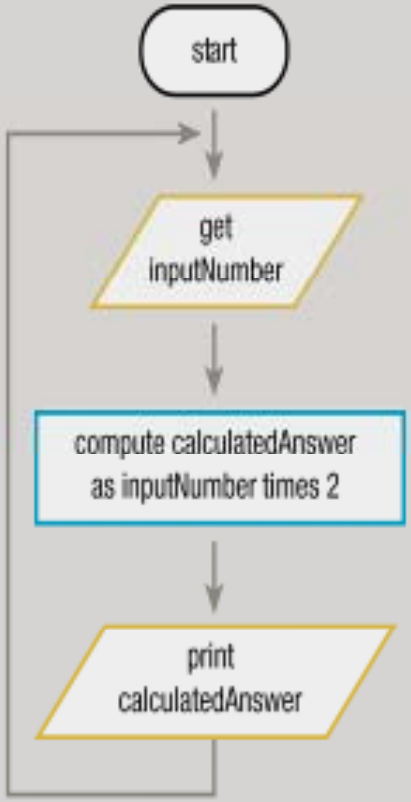
FIGURE 1-7: FLOWCHART AND PSEUDOCODE OF PROGRAM THAT DOUBLES A NUMBER

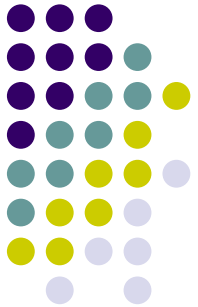


Using Flowchart Symbols and Pseudocode Statements (continued)

- Back-pointing arrows show statements that will be repeated

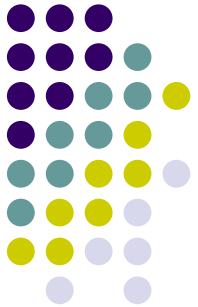
FIGURE 1-9: FLOWCHART OF INFINITE NUMBER-DOUBLING PROGRAM





Using and Naming Variables

- **Variable**: a memory location whose contents can vary
- Each programming language has it own rules for naming variables, including:
  - Legal characters
  - Maximum length
  - Use of upper or lower case
- Variable name must be a single word, but can be formed from several words
  - rate, interestRate, interest_rate
- In some languages you can define and use a variable in a single line of code, in others you need to define the variable before you can then store data in it

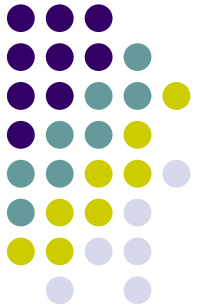


Using and Naming Variables (continued)

- Choose meaningful names for variables
  - Improves the readability and maintainability of code

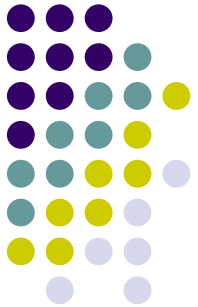
TABLE 1-1: VALID AND INVALID VARIABLE NAMES FOR AN EMPLOYEE'S LAST NAME

Suggested variable names for employee's last name	Comments
`employeeLastName`	Good
`employeeLast`	Good—most people would interpret `Last` as meaning `Last Name`
`empLast`	Good—`emp` is short for employee
`emlstnam`	Legal—but cryptic
`lastNameOfTheEmployeeInQuestion`	Legal—but awkward
`last name`	Not legal—embedded space
`employeelastname`	Legal—but hard to read without camel casing



Using and Naming Sessions

- **Session**: is like a variable, but is physically stored on a server environment
- Each web programming environment has its own ways of dealing with sessions
- Essentially, because a ‘client’ computer might be on the other side of the world from a ‘server’ computer (running a web application) they have no shared memory space
- Thus client-dependent data is stored in a session and read/updated between the client and server whenever the client moves from one page to another
- Naming conventions for sessions are similar to those for variables

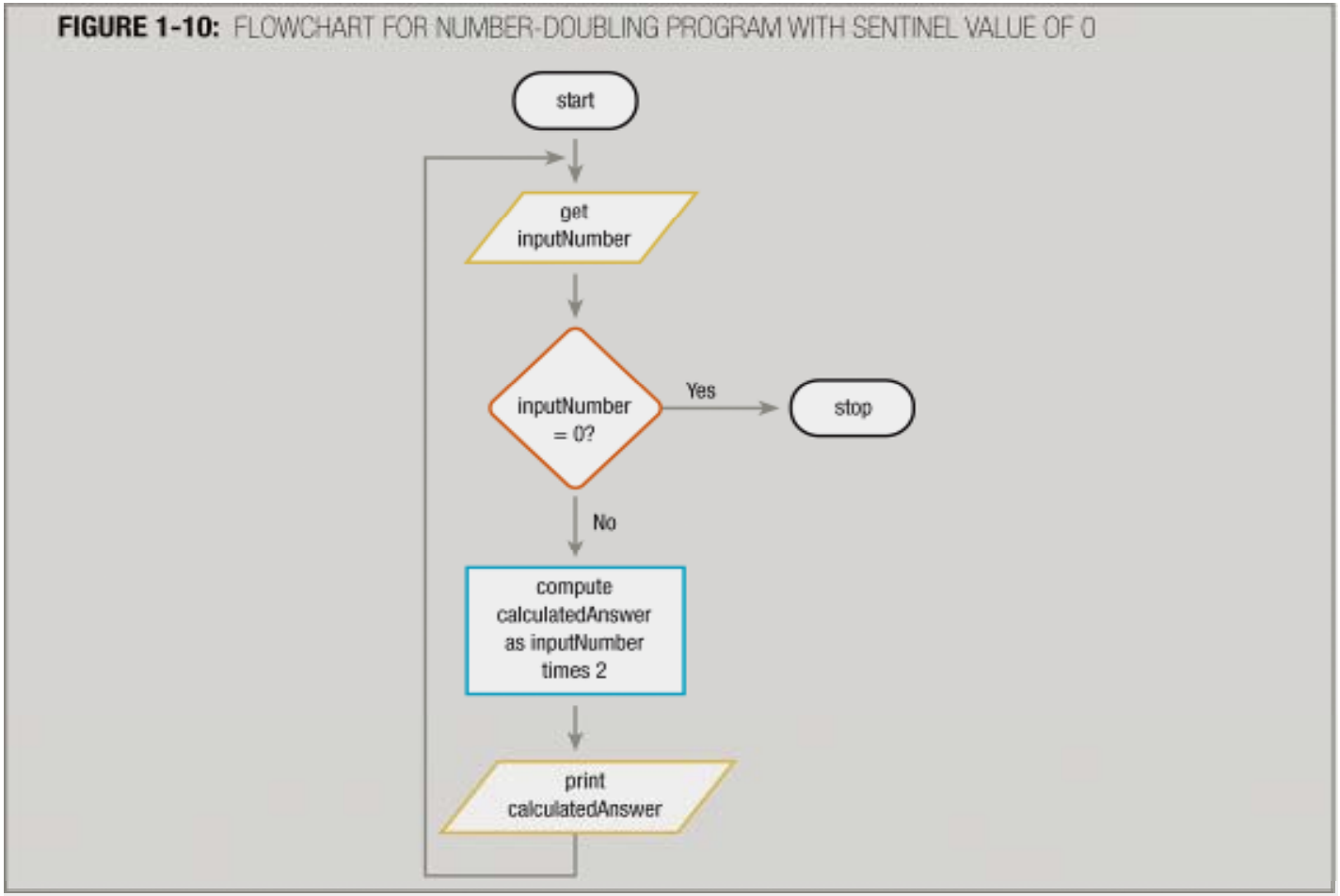


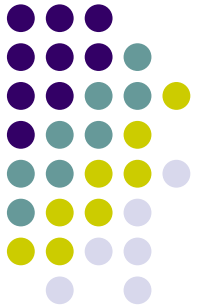
Ending a Program by Using Sentinel Values

- **Infinite loop**: a sequence of statements that repeats forever with no escape
- Avoid infinite loops by testing for a predetermined value that means "stop processing"
  - In well designed programs should not be necessary
  - However, it is easy to implement and gives piece of mind
- **Decision**: testing a value
- Flowchart **decision symbol**: a diamond shape, with two flowlines, one for Yes and one for No

Ending a Program by Using Sentinel Values (continued)

FIGURE 1-10: FLOWCHART FOR NUMBER-DOUBLING PROGRAM WITH SENTINEL VALUE OF 0



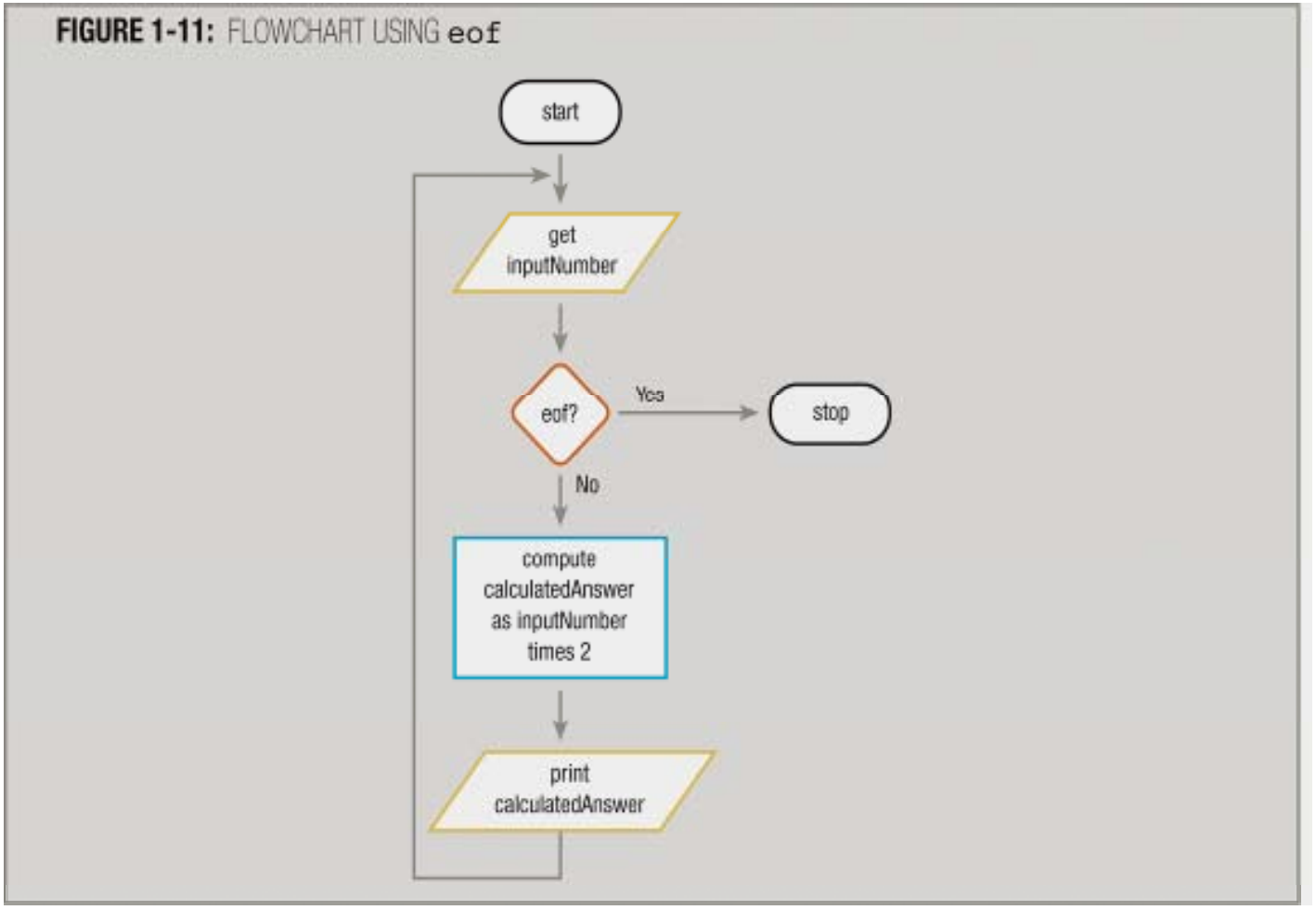


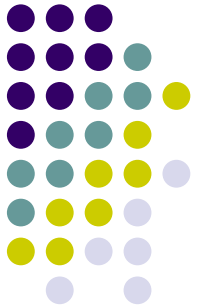
Ending a Program by Using Sentinel Values (continued)

- **Sentinel value** (or **dummy value**)
  - Does not represent real data
  - Signal to stop
  - Can be used with input from databases or from users
- **End-of-file** (**EOF**) **marker**:
  - Signal from a database recordset that marks the end of the data to be outputted
  - Usually used instead of a sentinel value for data input

Ending a Program by Using Sentinel Values (continued)

FIGURE 1-11: FLOWCHART USING `eof`

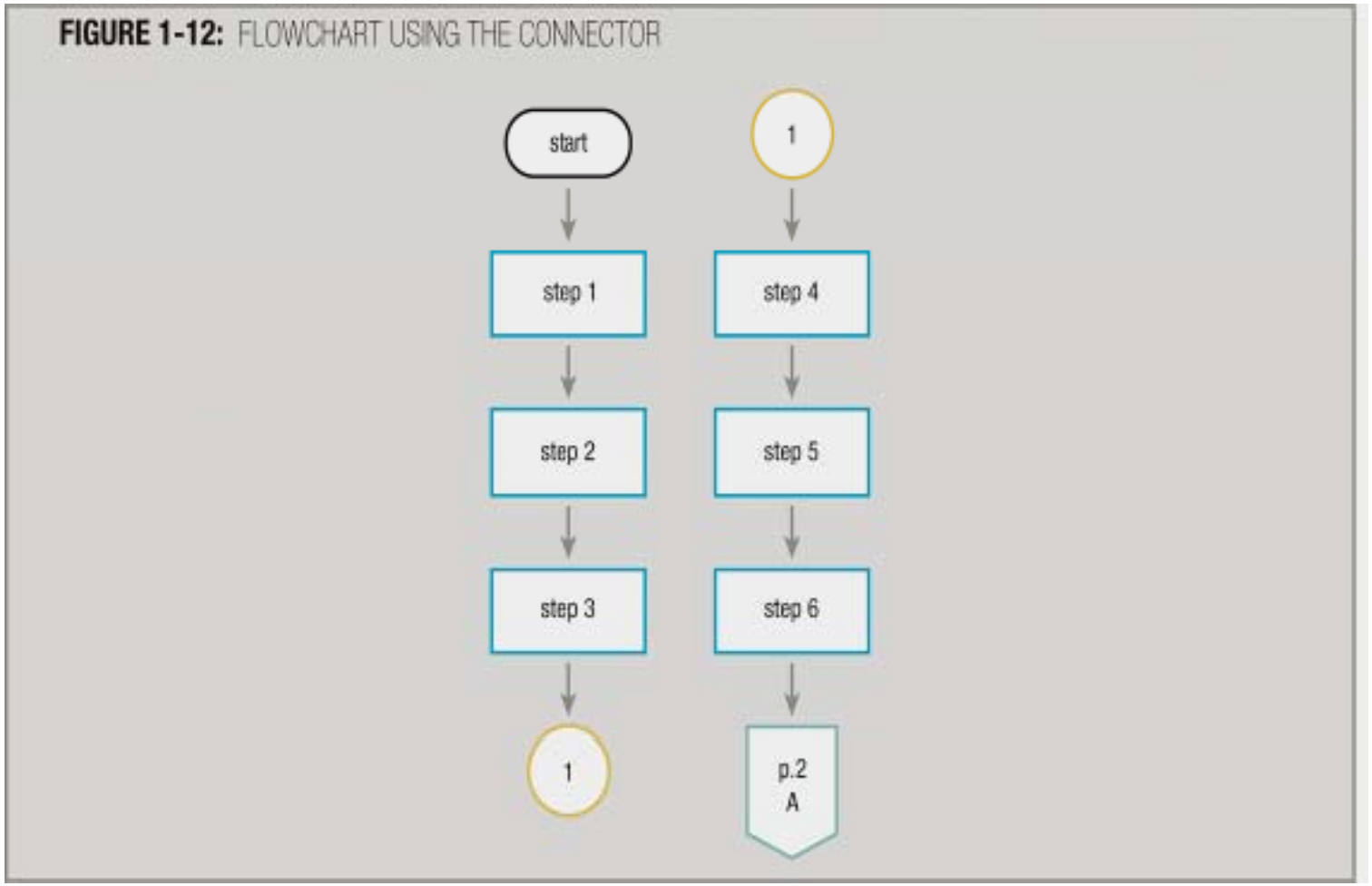


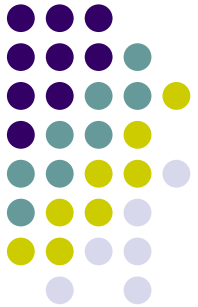


Using the Connector

- Flowchart **connector symbol**:
  - Marks a logic transfer to another location in the flowchart
  - Transfer location can be on the same page or on another page
  - **On-page symbol**: a circle with a number or letter to identify the matching transfer location
  - This is either accomplished by multiple ‘steps’ on a single page of the web app or by dynamically re-loading the same page for each step
  - **Off-page symbol**: a square with a pointed bottom, containing page number and a number of letter to identify the matching transfer location
  - In terms of web-dev, the off-page symbol typically means loading another page within the site

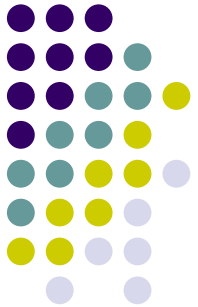
Using the Connector (continued)





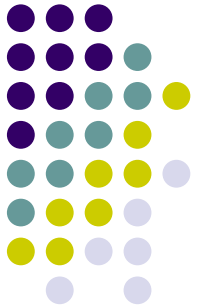
Assigning Values to Variables

- **Assignment statement**:
  - Assigns a value to a variable
  - Variable must appear on the left side, value on the right side of the assignment operator
  - Right side may be an expression that will be evaluated before storing the value in the variable
- **Assignment operator**: the equal sign (=) in most languages
- **Variable**:
  - Memory location: has an address and a value
  - Value (contents) is used for various operations
  - Most languages assign with one = and read (evaluate) variables with two ==



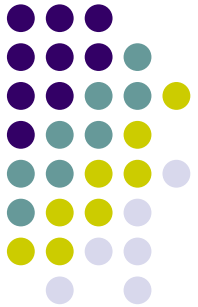
Understanding Data Types

- Two basic data types:
  - Text
  - Numeric
  - Datetime
- **Numeric data** stored by numeric variables
- **Text data** stored by string, text, or character variables
- **Datetime** while conceptually the same, are typically implemented in different formatting approaches across various languages and database systems
- **Constants**:
  - Values that do not change while the program is running
  - Have identifiers, and can be used like variables for calculations, but cannot be assigned new values



Understanding Data Types (continued)

- Some programming languages implement several numeric data types, such as:
  - **Integer**: whole numbers only
  - **Floating-point**: fractional numeric values with decimal points
  - **Boolean's:** such as 1's or 0's
- Character or string data is represented as characters enclosed in quotation marks
  - "x", "color"
- Data types must be used appropriately
- Some languages are relatively forgiving and will deal with numeric data automatically, other languages are extremely specific and will throw errors if incompatible data types are combined



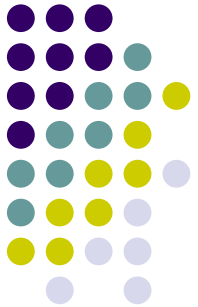
Understanding Data Types (continued)

TABLE 1-2: SOME EXAMPLES OF LEGAL AND ILLEGAL ASSIGNMENTS

Assume `lastName` *and* `firstName` *are character variables.*

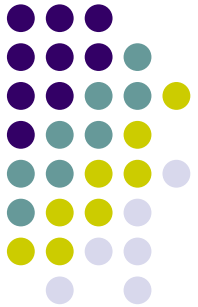
Assume `quizScore` *and* `homeworkScore` *are numeric variables.*

Examples of valid assignments	Examples of invalid assignments	Explanation of invalid examples
`lastName = "Parker"`	`lastName = Parker`	If `Parker` is the last name, it requires quotes. If `Parker` is a named string variable, this assignment would be allowed.
`firstName = "Laura"`	`"Parker" = lastName`	Value on left must be a variable name, not a constant
`lastName = firstName`	`lastName = quizScore`	The data types do not match
`quizScore = 86`	`homeworkScore = firstName`	The data types do not match
`homeworkScore = quizScore`	`homeworkScore = "92"`	The data types do not match
`homeworkScore = 92`	`quizScore = "zero"`	The data types do not match
`quizScore = homeworkScore + 25`	`firstName = 23`	The data types do not match
`homeworkScore = 3 * 10`	`100 = homeworkScore`	Value on left must be a variable name, not a constant



Understanding the Evolution of Programming Techniques (continued)

- Programming began in the 1940s, using memory addresses and machine code directly
- Higher level languages were developed to allow English-like instructions
- Older programs were “monolithic,” and ran from beginning to end
- Newer programs contain modules that can be combined to form programs
- Two major programming techniques:
  - Procedural programming
  - Object-oriented programming
- **Procedural programming**: focuses on the procedures that programmers create
- **Object-oriented programming**: focuses on objects that represent real-world things and their attributes and behaviors
- Both techniques employ reusable program modules



Web Readings

- How Web Applications are Revealing Our Future Through the Past by Daniel Nations – available at : http://webtrends.about.com/od/webapplications/a/future_webapps.htm
- Web Applications – A Guide to Success by Caesar Fernandes – Available at : http://articles.sitepoint.com/article/development-guide-success