HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

CSB4404 – PROGRAMMING PARADIGMS

B.Tech – VII Semester

Lecture 4-9

Dr. Muthukumaran M
Associate professor
School of Computing Sciences,
Department of Computer Science and Engineering

Language Categories

Programming languages are often categorized into four bins:

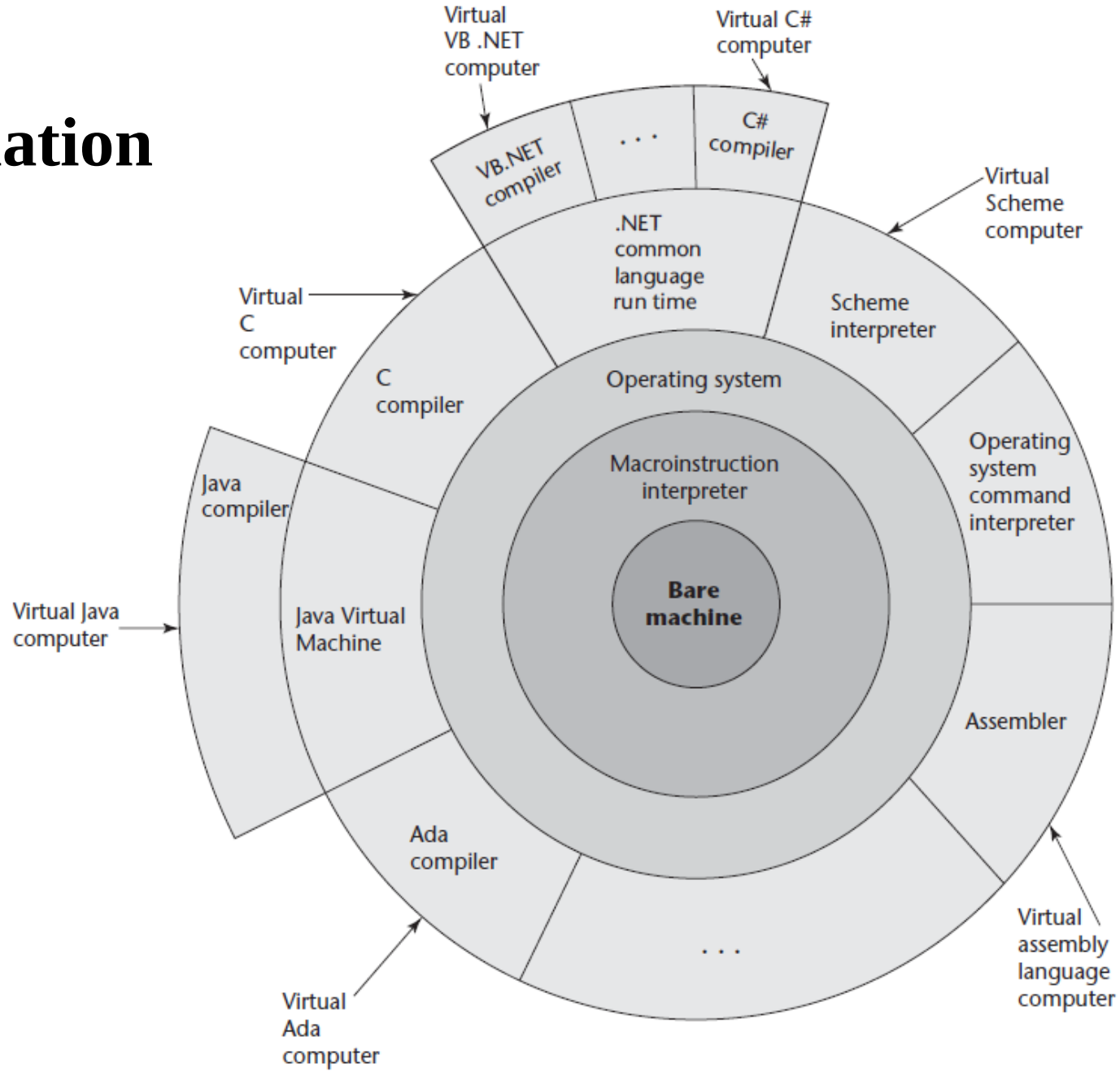
- Imperative
- Functional
- Logic
- object oriented.

Although the object- oriented software development paradigm differs significantly from the procedure- oriented paradigm usually used with imperative languages.

- For example, the expressions, assignment statements, and control statements of C and Java are nearly identical.
- (On the other hand, the arrays, subprograms, and semantics of Java are very different from those of C.)
- Some authors refer to scripting languages as a separate category of programming languages.
- A logic programming language is an example of a rule-based language.

Implementation methods

Compilation



Procedural languages:
FORTRAN, BASIC, C, COBOL, ALGOL 68, PL/I

FORTRAN (FORmula TRANslation)

(FORTRAN 0 - 1954 - not implemented)

- Designed for the new IBM 704, which had index registers and floating point hardware

- Environment of development:

1. Computers were small and unreliable
2. Applications were scientific
3. No programming methodology or tools
4. Machine efficiency was most important

- Impact of environment on design

1. No need for dynamic storage
2. Need good array handling and counting loops
3. No string handling, decimal arithmetic, or powerful input/output (commercial stuff)

- First implemented version of FORTRAN

- Names could have up to six characters
- Posttest counting loop (DO)
- Formatted i/o
- User-defined subprograms
- Three-way selection statement (arithmetic IF)
- No data typing statements
- No separate compilation
- Compiler released in April 1957, after 18 worker/ years of effort
- Programs larger than 400 lines rarely compiledcorrectly, mainly due to poor reliability of the 704
- Code was very fast
- Quickly became widely used

FORTRAN II - 1958

- Independent compilation
- Fix the bugs

FORTRAN IV - 1960-62

- Explicit type declarations
- Logical selection statement
- Subprogram names could be parameters
- ANSI standard in 1966

FORTRAN 77 - 1978

- Character string handling
- Logical loop control statement
- IF-THEN-ELSE statement

FORTRAN 90 - 1990

- Modules
- Dynamic arrays
- Pointers
- Recursion
- CASE statement
- Parameter type checking

FORTRAN Evaluation

- Dramatically changed forever the way computers are used

BASIC

BASIC - 1964

- Designed by Kemeny & Kurtz at Dartmouth
- **Design Goals:**
  - Easy to learn and use for non-science students
  - Must be ”pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
  - Current popular dialects: QuickBASIC and Visual BASIC

COBOL (COmmon Business Oriented Language)

COBOL - 1960

- Environment of development:

- UNIVAC was beginning to use FLOW-MATIC
- USAF was beginning to use AIMACO
- IBM was developing COMTRAN

- Based on FLOW-MATIC

- FLOW-MATIC features:
- Names up to 12 characters, with embedded hyphens
- English names for arithmetic operators
- Data and code were completely separate
- Verbs were first word in every statement

- First Design Meeting - May 1959

- Design goals:

    1. Must look like simple English
    2. Must be easy to use, even if that means it will be less powerful
    3. Must broaden the base of computer users
    4. Must not be biased by current compiler problems

- Design committee were all from computer manufacturers and DoD (Department of Defense) branches

- Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

- Contributions:

- First macro facility in a high-level language
- Hierarchical data structures (records)
- Nested selection statements
- Long names (up to 30 characters), with hyphens
- Data Division

- Comments:

- First language required by DoD; would have failed without DoD
- Still the most widely used business applications language

ALGOL (ALGOrithmic Language)

ALGOL 58 - 1958

- Environment of development:

1. FORTRAN had (barely) arrived for IBM 70x
2. Many other languages were being developed, all for specific machines
3. No portable language; all were machine- dependent
4. No universal language for communicating algorithms

- ACM and GAMM met for four days for design

- Goals of the language:

1. Close to mathematical notation
2. Good for describing algorithms
3. Must be translatable to machine code

- Language Features:

- Concept of type was formalized
- Names could have any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (begin ... end)
- Semicolon as a statement separator
- Assignment operator was :=
- if had an else-if clause

- Comments:

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid-1959

ALGOL 60 - 1960

- Modified ALGOL 58 at 6-day meeting in Paris
- New Features:
- Block structure (local scope)
- Two parameter passing methods
- Subprogram recursion
- Stack-dynamic arrays
- Still no i/o and no string handling

- **Successes:**
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
  - First machine-independent language
  - First language whose syntax was formally defined (BNF)
  - Failure:
  - Never widely used, especially in U.S.

Reasons:

1. No i/o and the character set made programs nonportable
3. Too flexible--hard to implement
4. Intrenchment of FORTRAN
5. Formal syntax description
6. Lack of support of IBM

ALGOL 68 - 1968

- From the continued development of ALGOL 60, but it is not a superset of that language
- Design is based on the concept of orthogonality
- Contributions:

1. User-defined data structures
2. Reference types
3. Dynamic arrays (called flex arrays)

- Comments:

- Had even less usage than ALGOL 60
- Had strong influence on subsequent languages, especially Pascal, C, and Ada

PL/I (Procedural, Imperative computer programming language)

PL/I - 1965

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)

1. Scientific computing

- IBM 1620 and 7090 computers
- FORTRAN
- SHARE user group

2. Business computing

- IBM 1401, 7080 computers
- COBOL
- GUIDE user group

- By 1963, however,

- Scientific users began to need more elaborate i/o, like COBOL had; Business users began to need fl. pt. and arrays (MIS)
- It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly

- The obvious solution:

1. Build a new computer to do both kinds of applications
2. Design a new language to do both kinds of applications

- PL/I contributions:

1. First unit-level concurrency
2. First exception handling
3. Switch-selectable recursion
4. First pointer data type
5. First array cross sections

- Comments:

- Many new features were poorly designed
- Too large and too complex
- Was (and still is) actually used for both scientific and business applications

C - 1972

- Designed for systems programming
(at Bell Labs by Dennis Richie)
- Evolved primarily from B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the founder of the c language.

It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL.

Object oriented Languages:
ADA, SIMULA, Small Talk, C++, Java, C#

Ada - 1983 (began in mid-1970s)

- Huge design effort, involving hundreds of people, much money, and about eight years

- Contributions:

1. Packages - support for data abstraction
2. Exception handling - elaborate
3. Generic program units
4. Concurrency - through the tasking model

- Comments:

- Competitive design
- Included all that was then known about software engineering and language design
- First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

- Ada 95 (began in 1988)

- Support for OOP through type derivation
- Better control mechanisms for shared data

(new concurrency features)

- More flexible libraries

SIMULA 67 - 1967

- Designed primarily for system simulation

(in Norway by Nygaard and Dahl)

- Based on ALGOL 60 and SIMULA I

- **Primary Contribution:**

    - Coroutines - a kind of subprogram

    - Implemented in a structure called a class

    - Classes are the basis for data abstraction

    - Classes are structures that include both local data and functionality

Smalltalk - 1972-1980

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg

- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic type binding)

- Pioneered the graphical user interface everyone now uses

C++ - 1985

- Developed at Bell Labs by Stroustrup

- Evolved from C and SIMULA 67

- Facilities for object-oriented programming, taken partially from SIMULA 67, were added to C

- Also has exception handling

- A large and complex language, in part because it supports both procedural and OO programming

- Rapidly grew in popularity, along with OOP

- ANSI standard approved in November, 1997

Java (1995)

- Developed at Sun in the early 1990s
- Based on C++
- Significantly simplified
- Supports only OOP
- Has references, but not pointers
- Includes support for applets and a form of concurrency

Smalltalk - 1972-1980

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg

- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic type binding)

- Pioneered the graphical user interface everyone now uses

C# was developed by Microsoft within its .NET framework initiative and later approved as a standard by ECMA (ECMA-334) C# programming language is a general-purpose, OOPS based programming language.

C# development team was lead by "Anders Hejlsberg" in 2002.

C# programming language is one of the languages designed for the (CLI) Common Language Infrastructure.

First version of C# is 1.0 with .NET framework 1.0 and Visual Studio is 2002.

C# language history

- **C# 1.0/1.2** (2002) – Modern, object-oriented, type safe programming language
- **C# 2.0 (**2005) – Generics, partial classes, anonymous types, iterators, nullable types, static classes, delegate interface
- **C# 3.0** (2007) – Implicit types, object/collection initializers, auto-implemented properties, extension methods, lambda expressions, expression trees, partial methods
- **C# 4.0** (2010) – Dynamic binding, named and optional arguments, generic covariance and contravariance
- **C# 5.0** (2013) – Async methods, caller info attributes, tuples
- **C# 6.0** (2015) – Roslyn, await in catch/finally, auto property initializer, string interpolation, nameof operator, dictionary initializers
- **C# 7.0** (2016) – Tuples, pattern matching, decomposition, improved out variables, ref returns, local functions, literal improvements

Logic Programming: Prolog

Prolog - 1972

- Developed at the University of Aix-Marseille, by Comerauer and Roussel, with some help from Kowalski at the University of Edinburgh

- Based on formal logic

- Non-procedural

- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries

Overview

- Prolog is a declarative logic programming language.
- It was created by Alain Colmerauer and Robert Kowalski around 1972 as an alternative to the American-dominated Lisp programming languages.
- It is an attempt to make a programming language that enables the expression of logic instead of carefully specified instructions on the computer.
- Prolog has got superb pattern matching mechanism as well as good memory management.
- It is very commonly used in artificial intelligence applications.

- The execution of a Prolog program is equivalent to searching the tree of possibilities and determining the objects which satisfy the given rules. This is equivalent to proof by proposition.
- Prolog has an excellent backtrack mechanism, so when exploring the tree it can always return to another branch if the present branch does not contain the answer.
- This approach is equivalent to depth first search which is known as an efficient algorithm for traversing trees with a large number of nodes.

- Prolog is a conversational language, meaning the user and the computer have a 'conversation’.
- Firstly, the user specifies the objects and the relationships that exist between these objects.
- Prolog can then answer the questions about this set of data.
- Prolog has a very compact syntax.
- From the developers point of view the difficulty of programming in Prolog, is the vagueness in human thinking.
- Prolog makes the coding easier as the syntax is very short and precise.

The Fundamentals of Prolog

Prolog is made up of facts and rules. Facts determine the nature of an object. Rules are properties that can be derived from the relations.

Facts:

The object determined by a fact, can belong to a certain category. For example "John is a male", can be coded as male(john). Facts also represent relationships which are true in a given domain. These facts can involve more that one object. For example the statement parent(john, tom). This means that there exists a relation between Tom and John called parent. The order of the clauses (objects) in the relation matters. This statement means that John is a parent of Tom, but the reverse is not true.

Rules:

Rules are properties that can be derived from the relations. A new relation can be concluded from the relations that already exist. For example a new relation John is Tom's father must imply that John is a parent of Tom and John is a male, provided that the relations "parent" and "male" are already defined.

This can be coded as follows; father(john,tom) :- male(john), parent(john,tom). The symbol ":-" means "if". In this context if the RHS is true the LHS is also true.

The comma in between represents the conjunction of computations. It means that both relations male and parent have to be true in order to conclude relation father.

Variables:

A variable represents an object in the circumstances where name is irrelevant. What matters, is whether the object that meets the criteria exists or not.

This means that questions such as Who are John's children? can be asked. For example the following code: ?- parent(john,X) would mean is there an X such that John is a parent of X.

If X exists, Prolog will return one of the possible values of X, otherwise the answer will be no. If asked, Prolog can also return all other possible answers, if they exist.

Prolog can distinguish between variables and the names of particular objects, since any name beginning with a capital letter is taken to be a variable.

Metaprogramming

A metaprogram is a program which takes other programs as data. Most declarative programming languages, especially Prolog, are powerful enough to possess other languages and programming paradigms. This is because Prolog implements symbol manipulation. This feature enables replacing a term or an argument with a predicate without loosing overall logical sense.

Functional Programming :LISP

Lisp is the second-oldest high-level programming language after Fortran and has changed a great deal since its early days, and a number of dialects have existed over its history. Today, the most widely known general-purpose Lisp dialects are Common Lisp and Scheme.

John McCarthy invented LISP in 1958, shortly after the development of FORTRAN. It was first implemented by Steve Russell on an IBM 704 computer.

It is particularly suitable for Artificial Intelligence programs, as it processes symbolic information effectively.

Common Lisp originated, during the 1980s and 1990s, in an attempt to unify the work of several implementation groups that were successors to Maclisp, like ZetaLisp and NIL (New Implementation of Lisp) etc.

It serves as a common language, which can be easily extended for specific implementation.

Programs written in Common LISP do not depend on machine-specific characteristics, such as word length etc.

Features of Common LISP

- It is machine-independent
- It uses iterative design methodology, and easy extensibility.
- It allows updating the programs dynamically.
- It provides high level debugging.
- It provides advanced object-oriented programming.
- It provides a convenient macro system.
- It provides wide-ranging data types like, objects, structures, lists, vectors, adjustable arrays, hash-tables, and symbols.
- It is expression-based.
- It provides an object-oriented condition system.
- It provides a complete I/O library.
- It provides extensive control structures.

Local Environment Setup

If you are still willing to set up your environment for Lisp programming language, you need the following two softwares available on your computer, (a) Text Editor and (b) The Lisp Executer.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

A Simple Program

Let us write an s-expression to find the sum of three numbers 7, 9 and 11. To do this, we can type at the interpreter prompt.

```
(+ 7 9 11)
```

LISP returns the result –

```
27
```

If you would like to run the same program as a compiled code, then create a LISP source code file named myprog.lisp and type the following code in it.

```
(write (+ 7 9 11))
```

When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is −

```
27
```

LISP Uses Prefix Notation

You might have noted that LISP uses prefix notation.

In the above program the + symbol works as the function name for the process of summation of the numbers.

In prefix notation, operators are written before their operands. For example, the expression,

a * (b + c) / d

will be written as –

(/ (* a (+ b c)) d)

Let us take another example, let us write code for converting Fahrenheit temp of 60o F to the centigrade scale −

The mathematical expression for this conversion will be −

(60 * 9 / 5) + 32
Create a source code file named main.lisp and type the following code in it.

(write(+ (* (/ 9 5) 60) 32))
When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is−

140

Evaluation of LISP Programs

Evaluation of LISP programs has two parts −

Translation of program text into Lisp objects by a reader program

Implementation of the semantics of the language in terms of these objects by an evaluator program

The evaluation process takes the following steps –

The reader translates the strings of characters to LISP objects or s-expressions.

The evaluator defines syntax of Lisp forms that are built from s-expressions. This second level of evaluation defines a syntax that determines which s-expressions are LISP forms.

The evaluator works as a function that takes a valid LISP form as an argument and returns a value. This is the reason why we put the LISP expression in parenthesis, because we are sending the entire expression/form to the evaluator as arguments.

Scripting languages

Overview

- Scripting languages have evolved over the past 35 years.
- Early scripting languages were used by putting a list of commands, called a script, in a file to be interpreted.
- The first of these languages, named *sh (for shell)*, began as a small collection of commands that were interpreted as calls to system subprograms that performed utility functions, such as file management and simple file filtering.
- To this were added variables, control flow statements, functions, and various other capabilities, and the result is a complete programming language.

- One of the most powerful and widely known of these is *ksh* (Bolsky and Korn, 1995), which was developed by David Korn at Bell Laboratories.
- Another scripting language is *awk,* developed by Al Aho, Brian Kernighan, and Peter Weinberger at Bell Laboratories (Aho et al., 1988).
- *awk* began as a report-generation language but later became a more general-purpose language

Origins and Characteristics of Perl

- The Perl language, developed by Larry Wall, was originally a combination of sh and awk.
- Perl has grown significantly since its beginnings and is now a powerful, although still somewhat primitive, programming language

Perl has a number of interesting features

- Variables in Perl are statically typed and implicitly declared.
- There are three distinctive namespaces for variables, denoted by the first character of the variables' names.
- All scalar variable names begin with dollar signs ($), all array names begin with at signs (@), and all hash names (hashes are briefly described below) begin with percent signs (%).
- This convention makes variable names in programs more readable than those of any other programming language.

- Perl includes a large number of implicit variables.
- Some of them are used to store Perl parameters, such as the particular form of newline character or characters that are used in the implementation.
- Implicit variables are commonly used as default parameters to built-in functions and default operands for some operators.
- The implicit variables have distinctive—although cryptic—names, such as $! and @_. The implicit variables' names, like the user-defined variable names, use the three namespaces, so $! is a scalar.

- Perl’s arrays have two characteristics that set them apart from the arrays of the common imperative languages.
- First, they have dynamic length, meaning that they can grow and shrink as needed during execution.
- Second, arrays can be sparse, meaning that there can be gaps between the elements.
- These gaps do not take space in memory, and the iteration statement used for arrays, foreach, iterates over the missing elements.

- Perl is a powerful, but somewhat dangerous, language.
- Its scalar type stores both strings and numbers, which are normally stored in double-precision floating- point form.
- Depending on the context, numbers may be coerced to strings and vice versa.
- If a string is used in numeric context and the string cannot be converted to a number, zero is used and there is no warning or error message error detection in array element access.

The following is an example of a Perl program:

```
Perl Example Program
# Input: An integer, $listlen, where $listlen is less
# than 100, followed by $listlen-integer values.
# Output: The number of input values that are greater
than
# the average of all input values.
($sum, $result) = (0, 0);
$listlen = <STDIN>;
if (($listlen > 0) && ($listlen < 100)) {
# Read input into an array and compute the sum
for ($counter = 0; $counter < $listlen; $counter++) {
$intlist[$counter] = <STDIN>;
} #- end of for (counter ...
# Compute the average
$average = $sum / $listlen;
# Count the input values that are > average
foreach $num (@intlist) {
if ($num > $average) { $result++; }
} #- end of foreach $num ...
# Print result
print "Number of values > average is: $result \n";
} #- end of if (($listlen ...
else {
print "Error--input list length is not legal \n";
}
```

Origins and Characteristics of JavaScript

- Use of the Web exploded in the mid-1990s after the first graphical browsers appeared.
- The need for computation associated with HTML documents, which by themselves are completely static, quickly became critical.
- Computation on the server side was made possible with the common gateway interface (CGI), which allowed HTML documents to request the execution of programs on the server, with the results of such computations returned to the browser in the form of HTML documents.
- Computation on the browser end became available with the advent of Java applets.
- Both of these approaches have now been replaced for the most part by newer technologies, primarily scripting languages.

- JavaScript was originally developed by Brendan Eich at Netscape. Its original
- name was Mocha. It was later renamed LiveScript.
- In late 1995, Live- Script became a joint venture of Netscape and Sun Microsystems and its name was changed to JavaScript.
- JavaScript has gone through extensive evolution, moving from version 1.0 to version 1.5 by adding many new features and capabilities.
- A language standard for JavaScript was developed in the late 1990s by the European Computer Manufacturers Association (ECMA) as ECMA- 262. This standard has also been approved by the International Standards Organization (ISO) as ISO-16262. Microsoft's version of JavaScript is named JScript .NET

- Although a JavaScript interpreter could be embedded in many different applications, its most common use is embedded in Web browsers.
- JavaScript code is embedded in HTML documents and interpreted by the browser when the documents are displayed.
- The primary uses of JavaScript in Web programming are to validate form input data and create dynamic HTML documents.

- One of the most important uses of JavaScript is for dynamically creating and modifying HTML documents.
- JavaScript defines an object hierarchy that matches a hierarchical model of an HTML document, which is defined by the Document Object Model.
- Elements of an HTML document are accessed through these objects, providing the basis for dynamic control of the elements of documents.

```
// example.js
//    Input:  An integer, listLen, where listLen is less
//            than 100, followed by listLen-numeric values
// Output:    The number of input values that are greater
//            than the average of all input values

var intList = new Array(99);
var listLen, counter, sum = 0, result = 0;

listLen = prompt (
        "Please type the length of the input list", "");
```

```
if ((listLen > 0) && (listLen < 100)) {

// Get the input and compute its sum
    for (counter = 0; counter < listLen; counter++) {
        intList[counter] = prompt (
                        "Please type the next number", "");
        sum += parseInt(intList[counter]);
    }

// Compute the average
    average = sum / listLen;

// Count the input values that are > average
    for (counter = 0; counter < listLen; counter++)
        if (intList[counter] > average) result++;

// Display the results
    document.write("Number of values > average is: ",
                   result, "<br />");
} else
    document.write(
        "Error - input list length is not legal <br />");
```

Origins and Characteristics of PHP

- PHP (Tatroe, et al., 2013) was developed by Rasmus Lerdorf, a member of the Apache Group, in 1994. His initial motivation was to provide a tool to help track visitors to his personal Web site.
- In 1995, he developed a package called Personal Home Page Tools, which became the first publicly distributed version of PHP. Originally, PHP was an abbreviation for Personal Home Page.
- Later, its user community began using the recursive name PHP: Hypertext Preprocessor, which subsequently forced the original name into obscurity.
- PHP is now developed, distributed, and supported as an open-source product. PHP processors are resident on most Web servers.

- PHP is an HTML-embedded server-side scripting language specifically designed for Web applications.
- PHP code is interpreted on the Web server when an HTML document in which it is embedded has been requested by a browser.
- PHP code usually produces HTML code as output, which replaces the PHP code in the HTML document. Therefore, a Web browser never sees PHP code.
- PHP is similar to JavaScript in its syntactic appearance, the dynamic nature of its strings and arrays, and its use of dynamic typing.
- PHP's arrays are a combination of JavaScript's arrays and Perl's hashes.

Origins and Characteristics of Python

- Python (Lutz, 2013) is a relatively recent object-oriented interpreted scripting language. Its initial design was by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands in the early 1990s.
- Its development is now being done by the Python Software Foundation.
- Python is being used for the same kinds of applications as Perl: system administration, CGI programming, and other relatively small computing tasks.
- Python is an open-source system and is available for most common computing platforms. The Python implementation is available at www.python.org, which also has extensive information regarding Python.

- Python is object oriented, includes the pattern-matching capabilities of Perl, and has exception handling.
- Garbage collection is used to reclaim objects when they are no longer needed.
- Support for CGI programming, and form processing in particular, is provided by the cgi module.
- Modules that support cookies, networking, and database access are also available.
- Python includes support for concurrency with its threads, as well as support for network programming with its sockets.
- It also has more support for functional programming than other nonfunctional programming languages.

Origins and Characteristics of Ruby

Ruby (Thomas et al., 2005) was designed by Yukihiro Matsumoto (aka Matz) in the early 1990s and released in 1996.

Since then it has continually evolved. The motivation for Ruby was dissatisfaction of its designer with Perl and Python.

Although both Perl and Python support object-oriented programming, neither is a pure object-oriented language, at least in the sense that each has primitive (nonobject) types and each supports functions.

The primary characterizing feature of Ruby is that it is a pure object-oriented language, just as is Smalltalk.

Every data value is an object and all operations are via method calls.

The operators in Ruby are only syntactic mechanisms to specify method calls for the corresponding operations.

Because they are methods, they can be redefined. All classes, predefined or user defined, can be subclassed.

- Both classes and objects in Ruby are dynamic in the sense that methods can be dynamically added to either.
- This means that both classes and objects can have different sets of methods at different times during execution.
- So, different instantiations of the same class can behave differently.
- Collections of methods, data, and constants can be included in the definition of a class.

Origins and Characteristics of Lua

- Lua was designed in the early 1990s by Roberto Ierusalimschy, Waldemar Celes, and Luis Henrique de Figueiredo at the Pontifical University of Rio de Janeiro in Brazil.
- It is a scripting language that supports procedural and functional programming with extensibility as one of its primary goals.
- Among the languages that influenced its design are Scheme, Icon, and Python.

- Lua is similar to JavaScript in that it does not support object-oriented programming but it was clearly influenced by it.
- Both have objects that play the role of both classes and objects and both have prototype inheritance rather than class inheritance.
- However, in Lua, the language can be extended to support object-oriented programming.
- Lua uses garbage collection for its objects, which are all heap allocated. It uses dynamic typing, as do most of the other scripting languages.

- As in Scheme, Lua's functions are first-class values. Also, Lua supports closures.
- These capabilities allow it to be used for functional programming.
- Also like Scheme, Lua has only a single data structure, although in Lua's case, it is the table.
- Lua's tables extend PHP's associate arrays, which subsume the arrays of traditional imperative languages.
- References to table elements can take the form of references to traditional arrays, associative arrays, or records.
- Because functions are first-class values, they can be stored in tables, and such tables can serve as namespaces.

- Lua can conveniently be used as a scripting language extension to other languages.
- Like early implementations of Java, Lua is translated to an intermediate code and interpreted.
- It easily can be embedded simply in other systems, in part because of the small size of its interpreter, which is only about 150K bytes.