

PROGRAMMING LANGUAGES

Revised 2/6/2012

/training/etc

The Art of Knowledge.

This Page Intentionally Left Blank

Software Development for Non-Programmers

Software Development for Non-Programmers.....	1
---	---

C

C Programming.....	2
Advanced C Programming.....	3

C++

C++ Programming.....	4
C++ For Non C Programmers.....	5
Advanced C++ Programming.....	6

Java Technologies

Java Programming.....	7
Advanced Java Programming.....	8
Java Message Service (JMS).....	9
SWING.....	10
Introduction to Spring Framework.....	11
Introduction to Hibernate.....	12
Struts Framework.....	13
Tomcat Administration.....	14
JavaServer Faces (JSF).....	15
Introduction to Maven 3.....	16
MVN - 201 Development Infrastructure Design.....	17
EJB 3.x Update for EJB 2.x Developers.....	18
Developing Java Enterprise Applications Using EJB3.....	19

Python

Introduction to Python 3.....	20
Advanced Python 3.....	21

Perl

Perl Programming.....	22
Advanced Perl Programming.....	23

Ruby

Ruby Programming.....	24
-----------------------	----

Groovy

Groovy Programming.....	25
-------------------------	----

This Page Intentionally Left Blank

Course Description:

Introduction to Programming has a wealth of knowledge that aspiring programmers need to master before tackling their first programming language. This course consists of topics such as: hardware and software components of a computer system, the CPU, registers, operating systems, processes, number systems, data type formats, conversions among number bases, logic and reasoning, flow charting, pseudo code, control structures, differentiation among programming languages, from machine language to assemblers and compilers, data structures, libraries, networking, web programming, and a host of other topics. After learning the fundamentals, students will be given a chance to show their knowledge by writing some simple to intermediate programs in the C programming language.

Who Should Attend:

This course is intended as a prerequisite for anybody who wishes to become a programmer or who needs to know key programmer issues and who does not have the necessary background for such an undertaking. It is also ideal for those who wish to begin a programming career or for those web designers who wish to add a programming component to their skills. This course is also intended for those tasked with managing programmers despite having no programming experience. Likewise, testers will gain invaluable experience from the subject matter in this course.

Prerequisites:

There are no prerequisites for this course.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- List the major components of a computer system.
- Differentiate between various kinds of computer storage.
- Distinguish between system software and application software.
- List the steps in the instruction/execution cycle.
- Differentiate between various kinds of programming language translators and programming paradigms.
- Convert from and to the following bases: binary, octal, decimal, and hexadecimal.
- Demonstrate how negative numbers are stored in memory.
- Demonstrate how various data types are stored in memory.
- State the difference between a program and a process.
- Use various graphical techniques, such as flow charting and pseudo code, to specify the logic of a program.
- Read BNF notation that describe correct grammar for a programming language.
- Write C programs that use conditionals and loops.
- Demonstrate the benefits of using functions.
- Understand the relationship between arguments and parameters.
- Effectively use arrays and strings in C.
- Build and use static libraries in C.
- Discern the various interfaces in launching programs.

Course Outline:**Introduction**

Introduction
What is a Computer System?
Input Units
Output Units
Memory
Central Processing Unit
Components of the CPU
Instruction Execution Cycle
Cache
Software
Functions of an Operating System
Loading the Operating System
Loading an Application Program
Timesharing

Programming Languages

Introduction
Machine Language
Assembly Language
Compilers
BNF
Language Classification
Procedural vs. Object Oriented
Static vs. Dynamic Typing
Scripting vs. Non-Scripting
Choosing a Programming Language

The Programming Cycle

The Software Development Cycle
The Programming Cycle - Overview
Edit
Compile
Execute

Number Systems and Data Types

Number Bases
Base 10

Base 2
Base 8 and Base 16
Conversion Between Number Bases
Relationships Between Number Bases
Data Types
Negative Numbers
Sign Magnitude
Complement Arithmetic
Floating-Point Values
String Data
Data Types in C

Programming Skills

Spoken Languages vs. Programming Languages
Programming Skills
Problem Solving
Mathematics
Flow Charting
Pseudo Code

Fundamentals of C - Part 1

C Data Types
Variables
printf
Arithmetic Operators
Control Flow - Decisions
Relational Operators
Control Flow - Loops
The while Loop
The for Loop
for Loop vs. while Loop

Fundamentals of C - Part 2

Simple Conditions
Compound Conditions
Operators
Truth Conditions
Logical and Operator

Logical or Operator
Logical not Operator
Loop Considerations
exit
break
continue

Using Functions in a C Program

Introduction
Abstraction
Functions
Arguments and Parameters
return Statement
Using a Function
Some Coding Techniques

Arrays

Arrays vs. Non-Array Variables
Arrays
Array Subscripts
Iterating Through an Array
Finding a Value in an Array
Finding the Largest Value in an Array

Appendix A: User Interfaces

Interfaces
Character Based - Command Line
Character Based - Interactive
Graphical User Interfaces
Client/Server Computing
The Client
The Server
Running the Client/Server Application
Web Based Applications

Appendix B: Creating Libraries

Creating Libraries
Using Libraries

Appendix C: Strings in C

Character Arrays
String Input Functions
How String Functions Work
Other String Functions
String Output Functions
Handling the End of File
Converting a String to a Number

Course Description:

This course provides students with a comprehensive study of the C programming language. Classroom lectures stress the strengths of C, which provide programmers with the means of writing efficient, maintainable, and portable code. The lectures are supplemented with non-trivial lab exercises.

Who Should Attend:

This course is for programmers who have had experience in any programming language or have been tasked with a C programming project, and other technical types including managers and customer support engineers who need to know C.

Prerequisites:

Students should have taken the Software Development for Non-Programmers course or have experience with a programming or an assembly language.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Write C programs that are non-trivial.
- Use the variety of data types appropriate to specific programming problems.
- Utilize the modular features of the language.
- Demonstrate efficiency and readability.
- Demonstrate the use of the various control flow constructs.
- Use arrays as part of the software solution.
- Utilize pointers to efficiently solve problems.
- Include the structure data type as part of the solution.
- Create their own data types.
- Use functions from the portable C library.

Course Outline:**Getting Started**

What is C?
Background
Sample Program
Components of a C Program
Examples
Data Types
Variables
Naming Conventions for C Variables
Printing and Initializing Variables
Array Examples
Compiling and Executing a C Program

Functions and Operators

Examples of C Functions
Functions
sum Invoked from main
Invoking Functions
Elementary Operators
The operator= Operators
Operators
The Conditional Operator
Increment and Decrement Examples
Increment and Decrement Operators

Control Flow Constructs

Examples of Expressions
if
if else
while
for
Endless Loops
do while
break and continue
switch
else if

The C Preprocessor

#define
Macros
#include
Conditional Compilation
#ifdef
#ifndef

Simple I/O

Character I/O
End of File
Simple I/O Examples
Simple I/O Redirection
I/O with Character Arrays

More on Functions

General

Function Declarations
Returning a Value or Not
Function Prototypes
Arguments and Parameters
Organization of C Source Files
Extended Example
The getline Function
The strcmp Function
The check Function
The atoi Function
The average Function
Summary

Bit Manipulation

Defining the Problem Space
A Programming Example
Bit Wise Operators
Bit Manipulation Functions
Circular Shifts

Strings

Fundamental Concepts
Aggregate Operations
String Functions

Higher Dimensional Arrays

Array Dimensions
An Array as an Argument to a Function
String Arrays

Separate Compilation

Compiling Over Several Files
Function Scope
File Scope
Program Scope
Local static
register and extern
Object Files
Libraries
The C Loader
Header Files

Pointers (Part 1)

Fundamental Concepts
Pointer Operators and Operations
Changing an Argument with a Function Call
Pointer Arithmetic
Array Traversal
String Functions with Pointers
Pointer Difference
Prototypes for String Parameters
Relationship Between an Array and a Pointer
The Pointer Notation *p++

Pointers (Part 2)

Dynamic Storage Allocation - malloc
Functions Returning a Pointer
Initialization of Pointers
gets - a Function Returning a Pointer
An Array of Character Pointers
Two Dimensional Arrays vs. Array of Pointers
Command Line Arguments
Pointers to Pointers
Practice with Pointers
Function Pointers

Structures

Fundamental Concepts
Describing a Structure
Creating Structures
Operations on Structures
Functions Returning Structures
Passing Structures to Functions
Pointers to Structures
Array of Structures
Functions Returning a Pointer to a Structure

Structure Related Items

typedef - New Name for an Existing Type
Bit Fields
unions
Non-Homogeneous Arrays
Enumerations

File I/O

System Calls vs. Library Calls
Opening Disk Files
fopen
I/O Library Functions
Copying a File
Character Input vs. Line Input
scanf
printf
fclose
Servicing Errors - errno.h
feof

Information About Files

The stat Function
File Existence
Telling Time - time and ctime
Telling Time - localtime

I/O With Structures

A Database Application
The menu Function

The fwrite Function
The create_db Function
The fread Function
The print_db Function
fseek
The retrieve_db Function
fflush and ftell

Useful Library Functions

strstr
strchr, strrchr
system
strtok
strspn, strcspn
Math Functions
Character Testing Functions
exit and atexit
signal
memcpy and memset
qsort
Binary Search - bsearch

Appendix A: C Language Programming

Important Header Files
printf Formats
C Reserved Words
Conversion
Precedence Chart

Course Description:

This course broadens the skills of a C language programmer by introducing sophisticated problem solving techniques including the advanced use of pointers, abstract data types, data structures, portability, and optimization techniques. Skills are reinforced by hands-on laboratory exercises.

Who Should Attend:

This course is for anybody who has been programming in C for at least six months or who is a skilled programmer and has taken a C introductory course.

Prerequisites:

Students should have at least six months of C programming experience.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Master the use of pointers in a wide variety of problems.
- Use sophisticated pointer techniques to solve problems involving advanced data structures such as lists, stacks, queues and trees.
- Choose from a wide variety of data structures to implement the most efficient solution to a problem.
- Apply the proper optimization technique to your C code.
- Apply many portability techniques to your C code.
- Use bit manipulation techniques for efficient solutions to problems.
- Write programs which emphasize modern program design techniques which emphasize code reuse.
- Write powerful C programs which make "calls" directly into the UNIX operating system through the use of system calls.
- Decrease development time in writing applications through a more thorough understanding of sophisticated concepts in C.

Course Outline:**A Review of C**

Data Types
Operators
Control Flow Constructs - if
Loops
switch
Derived Data Types
Arrays
Array vs. Pointer
Arrays and Pointers
Structures
Header File for a Structure
Use of Structures
Structure References
Structure Assignments
Unions
Bitfields
Enumerations

Functions

Function Fundamentals
Function Prototypes
Function Invocation and Definition
Subprogram Examples
Functions Returning a Value
Return Value Considerations
Recursive Functions
Evaluation of Function Arguments
Variable Number of Arguments
Scope of Variables
Storage Class Attributes
Initialization

Bit Manipulation

Characteristics of Bitwise
Problems
Defining the Problem Space
Bitwise Operators
Readability Aids
Assigning Bit Values
Writing Bitwise Functions
Circular Shifts
Character Information Array
Direct Lookup
Mapping With Bits
Radix Sort

Pointers

Common Pointer Constructions
Pointer Arithmetic
Binary Search
Command Line Arguments

The Environment Pointer
Changing a Pointer through a
Function Call
Processing Arrays With Pointers
Simulation Example
Simulating Higher Dimensional
Arrays
Two Dimensional Arrays
Complex Declarations
Pointers to Functions
Surrogate Sorting: A Pointer
Application
Sorting with Large Records

**Designing Data Types:
Structures**

Steps in Creating Data Types
Rationale For a New Data Type
The File `fraction.h`
Operations on the Fraction Data
Types
Implementation of the Functions
Example Program Using Fractions
Applications with Fractions
Set Notation Examples
Creating the Set Type
Set Representation Example
Set Representation
Set Function Implementations
A Program Which Uses The Set
Data Type

Data Structures

Potential List Elements
Lists - What Are They?
Problems With a List as an Array
Lists as Arrays
Benefits of Linking Elements
A List of Linked Elements
Defining the List Data Type
The List Data Type
Implementations of List Functions
A Simple Program With a List
Other Types of Lists
The Need for Other Kinds of Lists
Ordered Lists
The `rand` Function
Circular Lists
Circular List Code
Circular Lists Principles
Two Way Lists - Example

Two Way Lists
Two Way List - `print, order`
Two Way List - `main`
Structures for Networks
Networks
Hashing is Close to Direct Lookup
Hashing / Searching
The Hashing Algorithm
Hashing `main` Program
Example Code For Linear Collision
Linear Collision Handling
The `chain.h` File
Chaining Collision Handling
Chaining - `lookup` Function
Chaining Code - `main`
Stacks
Stack Representation
Solving Problems With Stacks
Picture of the Stack
Push and Pop Functions
The Calculator Driver Program
Queues
Queue Driver Program
Binary Trees
Traversing Trees
`tree.h`
Left-Root-Right Traversal
Tree Algorithms

**Optimization
Techniques**

Knowing When to Optimize
Where to Optimize
Examples of Macros
Macros
Knowing When to Initialize
Initialization
Modifying the Data Structure -
Example
Caching - Ready Access
Invariant Expressions
Logical Inefficiencies
Odds and Ends Examples
Odds and Ends

Portability

Different Kinds of Portability
Source Code Portability
Prototype Problems
Portability with Functions
Problems with `ints`

Arithmetic Data Types
Problems with Bits
Bit Manipulation
Portable Masks
Pointer Problems
ANSI vs. Non-ANSI Examples
ANSI vs Non-ANSI
Odds and Ends Examples
Odds and Ends

**Appendix A: Software
Tools**

The `cc` Command
Different C Compilers
Options - Examples
Compiler Options
Conditional Compile Examples
Conditional Compilation
Positing Assertions
The `assert` Macro
Libraries
Header File Support
Libraries
Graphics Coordinates
A Graphics Example
Examples of the Need for `make`
The `make` Command
An Example Makefile
The `make` Dependency Tree
SCCS Example
Source Code Control System
After a Revision Cycle
Source Code Control System

**Appendix B: Library
Functions**

Building Command Strings
`system`
`exit` and `atexit` Examples
`exit` and `atexit`
`signal`
Using `strtok`
`strtok`
`memcpy` and `memset`
`memcpy` - `memset`
Using `qsort`
`quicksort`
Binary Search Example
`bsearch`
`strxtr` Example
`strxtr`

`strchr` Example
`strchr`, `strchr`
Data Validation Example
`strspn`, `strcspn`

Appendix C: File Access

I/O From Applications Programs
System Calls vs. Library Calls
The `fopen` Function
Opening Disk Files
Table of Access Modes
Access Modes
Reasons for an `fopen` Failure
Errors In Opening Files
Example: Copying a File
I/O Library Calls
Character vs Line Examples
Character Input vs Line Input
Interpreting Input
Motivation for the `scanf` Function
`scanf` Examples
`scanf`
`sscanf` Example
`scanf` Variants
`fscanf` and `scanf` Example
`scanf` Variants
`sprintf` and `fprintf`
Examples
`printf` Variants
An `fclose` Example
Closing Files - `fclose`
Error Examples
Servicing Errors - `errno`
Servicing Errors - Example
Servicing Errors - `errno.h`
Application for Binary I/O
Binary I/O
The `main` Function - Code
The `main` Function
`create_db` Function - `fwrite`
`fwrite`
`print_db` Function - `fread`
`fread`
`retrieve_db` Function
`fseek`
`fflush` and `ftell` Example
`fflush` and `ftell`
Exercises

Course Description:

C++ is the object oriented superset of ANSI C. This course provides students with a comprehensive study of the C++ Programming Language. The course stresses the object paradigm including classes, inheritance, virtual functions, and templates in the development of C++ programs. Lab exercises reinforce the lectures.

Who Should Attend:

Anybody who has the need to write programs in the C++ language including programmers, engineers, scientists, or other technical support personnel will benefit from this course.

Prerequisites:

Students should have taken the Software Development for Non-Programmers and Introduction to C courses or have equivalent knowledge.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Explain how object-oriented software engineering enhances the software development process.
- Identify the major elements in an object-oriented programming language.
- Implement the concepts of data abstraction and encapsulation in the creation of abstract data types.
- Implement operator overloading.
- Use inheritance in C++.
- Select the proper class protection mechanism.
- Demonstrate the use of virtual functions to implement polymorphism.
- Write programs utilizing the I/O classes in C++.
- Understand some advanced features of C++ including templates, exceptions, and multiple inheritance.
- Compare the object vs the procedural approach to writing software.
- Use correct object oriented terminology.
- Define and use classes in a C++ program.
- Create and use abstract data types.
- Derive classes using inheritance in C++.
- Implement polymorphism by using virtual functions in a program.

Course Outline:**Perspective**

The Software Crisis
Design Techniques
Large Software Systems
Roots of Object Technology
What Is Object-Oriented Programming?
C++ and Object-Oriented Programming
Why C++?
Features of C++
Pros and Cons of C++

The Language of Object-Oriented

What Is an Object?
What Is a Class?
Encapsulation
Data Hiding
The Public Interface
Relationships Among Classes
Inheritance
Polymorphism
Object-Oriented Design

C vs. C++

Comments
Namespaces
Simple Output
Simple Input
Definitions Near to First Use
Function Prototypes
The inline Specifier
const
Structure Members
The Reference Type
Overloading Function Names
Default Parameters
The Scope Resolution Operator
Aggregates
Operators new and delete
The bool Data Type
The string Data Type

Fundamentals of Classes

Data Types
User Defined Data Types
Using the Class Concept
Defining a Class
public and private Access Levels
The Scope Resolution Operator ::

Using Class Objects Like Built-in Types
Scope
Constructors
Member Initialization Lists
Destructors
Array of Objects
Pointers
The this Pointer
Passing Objects to Functions
Returning Objects From Functions
static Class Members

Operator Overloading

Introduction
Rules for Operator Overloading
Rationale for Operator Overloading
Overloading Member Functions
Overloading Non-Member Functions
friend Functions
The Copy Constructor
The Assignment Operator
Overloading []
Overloading Increment and Decrement
Operators
const Objects and References

Composition of Classes

Relationships
Composition of Classes
The Point Class
The Line Class
Member Initialization Lists
An Application With Composition
The Copy Constructor under Composition
operator= under Composition

Inheritance

Introduction
Public Base Classes
The protected Access Level
Member Initialization Lists
What Isn't Inherited
Assignments Between Base and Derived
Objects
Compile-Time vs. Run-Time Binding
virtual Functions
Polymorphism
virtual Destructors
Pure virtual Functions

Abstract Base Classes
An Extended Inheritance Example

I/O in C++

The ostream Library
Predefined Streams
Overloading operator<<
Overloading operator>>
Manipulators
Stream States
Formatted I/O
Disk Files
Reading and Writing Objects

Advanced Topics

Template Functions
Template Classes
Multiple Inheritance
User-Defined Conversions
Data Structures
An Iterator Class
Exceptions

Introduction to the Standard Template Library

Introduction
The Standard Template Library
Design Goals
STL Components
Iterators
Example: vector
Example: list
Example: set
Example: map
Example: find
Example: merge
Example: accumulate
Function Objects
Adaptors

Appendix A: Introduction

Background
Environmental Considerations
A Sample C Program
Variables and Data Types
Arrays
Example of a Program Using an int Array
Components of a C Program
C Operators

Examples of the Operators
Control Structures
Functions
Function Prototypes
Simple I/O

Appendix B: More I/O in C

The printf Function
The scanf Function
The Preprocessor
Conditional Compilation
Avoiding Multiple Inclusion for the Same File

Appendix C: Aggregates in C

Data Types Revisited
Aggregate Types
Arrays
Structures
Structures and Functions
Bit Fields
Enumeration Types

Appendix D: Pointers in C

Fundamental Concepts
Pointer Operations
Using Pointers to Alter a Function Argument
Using Pointers for Array Traversal
Pointer Arithmetic
Sending an Array to a Function
Command Line Arguments
Pointers vs. Arrays
Sending an Aggregate to a Function
Summary of the Uses of Pointers

Appendix E: Bibliography

Bibliography

Course Description:

This course provides students with a comprehensive study of the C++ programming language while teaching those parts of C relevant to C++. Classroom lectures are supplemented with many hands-on exercises, which stress the following C++ topics: data abstraction, class design, operator overloading, inheritance, polymorphism and I/O.

Who Should Attend:

This course is designed primarily for those Cobol, Pascal, and Fortran programmers who wish to learn C+ without having to partake in a prior C Language programming course.

Prerequisites:

Experience with a programming language or an assembly language is required.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Use correct object oriented terminology.
- Compare and choose between the object and the procedural approach to writing software.
- Compare and select the appropriate I/O model from either C or C++.
- Define and use classes in a C++ program.
- Select the proper class protection mechanism.
- Create and use abstract data types.
- Implement operator overloading in user defined and off the shelf classes.
- Derive classes using inheritance in C++.
- Implement polymorphism by using virtual functions in a C++ program.
- Utilize the modular features of the C and C++ language.

Course Outline:**Introduction**

Background
Environmental considerations
A Sample C program
Variables and data types
Arrays
Components of a C program
C operators
Control structures
Functions
Function prototypes
Simple I/O

More I/O In C

The printf function
The scanf function
The preprocessor
Conditional compilation
Avoiding multiple inclusions for the same file

Aggregates In C

Data types revisited
Aggregate types
Arrays
Structures
Structures and functions
Bit fields
Enumeration types

Pointers In C

Fundamental Concepts
Pointer operations
Using pointers to alter a function argument
Using pointers for array traversal
Pointer arithmetic
Sending an array to a function
Pointers vs arrays
Sending an aggregate to a function
Summary of the uses of pointers

Perspective

The software crisis
Building software has been difficult
Design techniques
Large Software Systems
Roots of Object Orientation
What is OO programming?
C++ and OO programming
Why C++?
Features of C++
Pros and Cons of C++

The Language Of Object Orientation

What is an object?

What is a class?
Encapsulation
Data hiding
The public Interface
Relationships among Classes
Inheritance
Polymorphism
Object-Oriented Design

C vs C++

Comments
Namespaces
Performing Simple Output
Performing Simple Input
Definitions near to first Use
Function prototypes
The inline specifier
const
Structure Members
The reference type
Overloading function names
Default parameters
Scope resolution operator
Aggregates
Operators new and delete
The bool Data Type
The string Data Type

Fundamentals Of Classes

Data types
Abstract data types
Using the class concept
How to define a class
public and private access levels
Using class objects like a built in type
scope
scope resolution operator
Constructors
Member Initialization Lists
Destructors
Array of Objects
Pointers
The this pointer
Passing Objects to Functions
Returning Objects from Functions
Static class members

Operator Overloading

Introduction
Rules for Operator Overloading
Rationale for Operator Overloading
Overloading Member Functions
Overloading Non-Member Functions

friend functions
The Copy Constructor
Overloading the Assignment Operator
Overloading []
Overloading increment and decrement operators
const Objects & const references

Composition Of Classes

Relationships
Composition of Classes
The Point class
The Line class
Member Initialization Lists
An Application w/ composition
The Copy Constructor under Composition
Operator= under Composition

Inheritance

Introduction
Inheritance public base classes
Inheritance w/ public base classes
Member Initialization Lists
What isn't inherited
Assignments between base and derived Objects
Compile Time Binding vs. Run Time Binding
virtual functions
Polymorphism
virtual destructors
Pure virtual functions
Abstract base classes
An extended inheritance example

I/O In C++

The iostream library
Pre-defined streams
operator<<
Overloading << for User-Defined Classes
Overloading >> for User-Defined Classes
Manipulators
Stream states
Formatted i/o
Disk files
Internal transmission of data
Reading & Writing Objects

Advanced Topics

Template Functions
Template Classes
Multiple Inheritance
User-Defined Conversions
Data Structures
Iterators
Exceptions

Course Description:

This course broadens the skills of a C++ language programmer by examining sophisticated C++ concepts such as templates, exceptions, memory management, advanced inheritance issues, disambiguation of overloaded functions, private and protected inheritance, binary I/O and class libraries.

Who Should Attend:

This course is for anybody who has programmed in C++ and wishes to enhance their knowledge of the language.

Prerequisites:

Students should have completed an introductory C++ programming course or have equivalent knowledge.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Differentiate between global functions, friend functions and member functions.
- Code their own memory management routines by overloading operators new and delete.
- Write classes and functions with parameterized types.
- Understand and handle exceptions in C++ programs.
- Disambiguate data and functions using multiple inheritance.
- Understand the difference between various kinds of inheritance.
- Use pointers to class member functions.
- Understand the C++ mechanism to resolve overloaded functions.

Course Outline:**What You Should Already Know - A Review**

Rationale for a New Programming Language
The Language of Object-Oriented
A Typical C++ Class
Issues Regarding Member Functions vs. Non-Member Functions
friend or *non-friend*
Functions Returning References
Relationships
Initialization Lists
Inheritance In C++
Access Levels
Simple C++ I/O
The Many Uses of `const`

Parameterized Types - Templates

Templates
Overloading Functions
Template Functions
Specializing a Template Function
Disambiguation Under Specialization
Template Classes
Instantiating a Template Class Object
Rules for Template Classes
A Non-Member Function with a Template Argument
Friends of Template Classes
Templates with Multiple Type Parameters
Comments Regarding Templates

Relationships of all Kinds

Uses of Member Initialization Lists
Member Initialization Lists Under Composition
Initialization Lists Under Inheritance
Initialization Lists With Multiple Inheritance
Initialization Lists with Multiple Inheritance and Composition
Efficiency
`operator=` and Composition
Constructors and Composition
What is not Inherited?
`operator=`, Construction, and Inheritance
Public Inheritance
virtual Functions
A Shape Class Hierarchy
Polymorphism
Pure Virtual Functions
Abstract Base Classes
Private Inheritance
using Relationships
Associations

Multiple Inheritance (MI)

Multiple Inheritance
Ambiguities

virtual Base Classes
The Dominance Rule
Member Initialization Lists
`operator=`

Data Structures

Introduction
A Simple List
Layering Type-safe Classes Upon List
A template List Class
Iterators
A template Iterator
Stack and Queue Classes
Templates and Inheritance

Function Pointers

Why Have Function Pointers?
Passing Functions as Arguments
Registering Functions
Function Pointers in C++
Callback Functions
A Class with a Callback Object
Registration of Exception Handlers

Exceptions

What Are Exceptions?
Traditional Approaches to Error Handling
`try`, `catch`, and `throw`
A Simple Exception Handler
Multiple `catch` Blocks
The Exception Specification List
Rethrowing an Exception
Cleanup
Exception Matching
Inheritance and Exceptions
Resource Allocation
Constructors and Exceptions
Destructors and Exceptions
Catch by Reference
Standard Exceptions

The C++ Standard Template Library

Introduction
The Standard Template Library
Design Goals
STL Components
Iterators
Example: `vector`
Example: `list`
Example: `set`
Example: `map`
Example: `find`
Example: `merge`
Example: `accumulate`
Function Objects

Adaptors

Disambiguation

Conversion
`int` Conversion
`float + double` Conversions
Arithmetic and Pointer Conversion
Inheritance Based Conversion
Overloaded Functions
Exact Match
Match with Promotion
Match with Standard Conversion
User Defined Conversion
Constructors as Conversion Operators
Ambiguities

I/O

Introduction
Manipulators
Writing Your Own Manipulators
Overloading the I/O Operators
Disk Files
Reading and Writing Objects
Internal Transmission of Data
A Spell Checker
Handling Streams in the Constructor and Destructor
Treating a File as an Array
Example

Miscellaneous Topics

Namespaces
Use Counts
Reference Counts
RTTI
Casts
Having a Limited Number of Objects
Smart Pointers

Course Description:

This course teaches students how to develop Java applications. Topics covered include the Java programming language syntax, OO programming using Java, exception handling, file input/output, threads, collection classes, and networking. Students will develop and test Java applications (typically) using Eclipse. This course is a pre-requisite to all Application Server courses, and speciality Java Technology courses such as Struts, Spring, and Hibernate.

Who Should Attend:

This course is designed for applications programmers and designers planning to develop applications using the Java Virtual Machine.

Prerequisites:

Students should have taken the Software Development for Non-Programmers course or have programmed in at least one programming language - preferably C or C++. Some familiarity with Object Oriented Programming is desired but not required.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Compile and run a Java application.
- Understand the role of the Java Virtual Machine in achieving platform independence.
- Navigate through the API docs.
- Use the Object Oriented paradigm in Java programs.
- Understand the division of classes into Java packages.
- Use Exceptions to handle run time errors.
- Select the proper I/O class among those provided by the JDK.
- Use threads in order to create more efficient Java programs.

Course Outline:**Introduction**

What is Java?
History
Versioning
The Java Virtual Machine
Writing a Java Program
Packages
Simple Java Programs

Language Components

Primitive Data Types
Comments
The for Statement
The if Statement
The while and do while Statements
The switch Statement
The break Statement
The continue Statement
Operators
Casts and Conversions
Keywords

Object-Oriented Programming

Defining New Data Types
Constructors
The String Class
String Literals
Documentation
Packages
The StringBuffer Class
Naming Conventions
The Date Class
The import Statement
Deprecation
The StringTokenizer Class
The DecimalFormat Class

Methods

Introduction
Method Signatures
Arguments and Parameters
Passing Objects to Methods
Method Overloading
Static Methods
The Math Class
The System Class
Wrapper Classes

Arrays

Introduction
Processing Arrays
Copying Arrays
Passing Arrays to Methods
Arrays of Objects

The Arrays Class
Command Line Arguments
Multidimensional Arrays

Encapsulation

Introduction
Constructors
The this Reference
Data Hiding
public and private Members
Access Levels
Composition
Static Data Members

Inheritance & Polymorphism

Introduction
A Simple Example
The Object Class
Method Overriding
Polymorphism
Additional Inheritance Examples
Other Inheritance Issues

Abstract Classes and Interfaces

Introduction
Abstract Classes
Abstract Class Example
Extending an Abstract Class
Interfaces

Exceptions

Introduction
Exception Handling
The Exception Hierarchy
Checked Exceptions
Advertising Exceptions with throws
Developing Your Own Exception Classes
The finally Block

Input and Output in Java

Introduction
The File Class
Standard Streams
Keyboard Input
File I/O Using Byte Streams
Character Streams
File I/O Using Character Streams
Buffered Streams
File I/O Using a Buffered Stream
Keyboard Input Using a Buffered Stream
Writing Text Files

Threads

Threads vs. Processes

Creating Threads by Extending Thread
Creating Threads by Implementing Runnable
Advantages of Using Threads
Daemon Threads
Thread States
Thread Problems
Synchronization

Collections

Introduction
Vectors
Hashtables
Enumerations
Properties
Collection Framework Hierarchy
Lists
Sets
Maps
The Collections Class

Networking

Networking Fundamentals
The Client/Server Model
InetAddress
URLs
Sockets
A Time-of-Day Client
Writing Servers
Client/Server Example

Course Description:

Advanced Java is a comprehensive study of many advanced Java topics. These include assertions, collection classes, searching and sorting, regular expressions, logging, bit manipulation, serialization, threads, networking with sockets, Remote Method Invocation, and Java Database Connectivity.

Who Should Attend:

This course is intended for Java programmers who wish to write programs using many of the advanced Java features.

Prerequisites:

Students should have completed a beginning Java course or have programmed in Java for at least three to six months.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Document and package a Java application.
- Use many of the new enhancements added to the Java API.
- Use assertions to write robust Java code.
- Use regular expressions for efficient pattern matching.
- Choose appropriate data structures from the Java Collection API.
- Sort and search arrays and lists using a variety of techniques.
- Capture configuration and debugging information using the Java Logging APIs.
- Use Generics to create type safe collections.
- Serialize Java objects.
- Use features of the new I/O API.
- Write TCP/IP Client Server applications using sockets.
- Write multi-threaded Java applications.
- Execute methods on a remote object using RMI.
- Perform database queries and updates using JDBC.

Course Outline:**Review of Java Fundamentals**

The Java Environment
Data Types
The String Class
The StringBuffer Class
Arrays
Passing Data Types to a Method
Constructors and Initialization
Inheritance
Abstract Classes
Interfaces
Static Data, Methods, and Blocks
Wrapper Classes
I/O

Packaging and Distributing a Java Application

Packages
Managing Source and Class Files
The javadoc Utility
Documenting Classes and Interfaces
Documenting Fields
Documenting Constructors and Methods
Running the javadoc Utility
jar Files
The Manifest File
Bundling and Using Jar-Packaged Resources

Miscellaneous Enhancements

Enhanced For Loop
Autoboxing and Auto-Unboxing
Static Imports
varArgs
Typesafe Enums
Formatted Strings
Format Specifier Syntax
Format Specifier Conversions
Format Specifier Flags
Formatted Integers Example
Formatted Floating Points Example
Formatted Strings Example
Formatted Dates Example
Complex Formatted Example

Assertions

Introduction
Assertion Syntax
Compiling with Assertions
Enabling and Disabling Assertions

Assertion Usage

Regular Expressions

Regular Expressions
String Literals
Character Classes
Quantifiers
Capturing Groups and Backreferences
Boundary Matchers
Pattern and Matcher

The Java Collection Classes

Introduction
The Arrays Class
Searching and Sorting Arrays of Primitives
Sorting Arrays of Objects
The Comparable and Comparator Interfaces
Sorting - Using Comparable
Sorting - Using Comparator
Collections
Lists and Sets
Iterators
Lists and Iterators Example
Maps
Maps and Iterators Example
The Collections Class
Rules of Thumb

Generics

Introduction
Defining Simple Generics
Generics and Subtyping
Wildcards
Bounded Wildcards
Generic Methods

Advanced I/O

Introduction
Basic File I/O Example
Buffered I/O
The Console Class
Object Serialization
Serialization Issues
Compressed Files
Zip File Example
Writing Your Own I/O Classes
Property Files
The Preferences Class

Enhanced I/O

Introduction
Channels
Buffers
Typed Buffers
Direct Buffers

Logging API

Introduction
Loggers
Logger Levels
Logger Handlers
Specifying Handlers and Formatters
Configuring Handlers
LogManager

Networking

Networking Fundamentals
The Client/Server Model
InetAddress
URLs
Sockets
A Time-of-Day Client
Writing Servers
Client/Server Example

Threads and Concurrency

Review of Fundamentals
Creating Threads by Extending Thread
Creating Threads by Implementing Runnable
Advantages of Using Threads
Daemon Threads
Thread States
Thread Problems
Synchronization
Performance Issues

Remote Method Invocation (RMI)

Introduction
RMI Architecture
The Remote Interface
The Remote Object
Writing the Server
The RMI Compiler
Writing the Client
Remote Method Arguments and Return Values
Dynamic Loading of Stub Classes
Remote RMI Client Example
Running the Remote RMI Client Example

Java Database Connectivity (JDBC)

Introduction
Relational Databases
Structured Query Language
A Sample Program
Transactions
Meta Data

Course Description:

This course introduces Java developers to the Java Message Service (JMS). Basic messaging concepts and the JMS API programming model are covered along with details of the JMS message format. Tuning mechanisms for improving JMS reliability are discussed and demonstrated. WebLogic Server is used as the JMS Provider for examples and programming exercises.

Who Should Attend:

All developers wishing to learn about the Java Messaging Service should attend this course.

Prerequisites:

Java programming experience with J2EE technology and/or WebLogic is required.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Compare point-to-point and publish subscribe messaging.
- Understand the JMS programming model, including connection factories, connections, sessions, consumers, producers, and destinations.
- Use different JMS message types.
- Use JMS message header fields and properties
- Tune JMS reliability features such as message persistence, message priority, and message expiration.
- Create durable subscribers and temporary destinations.
- Commit and rollback JMS transactions.
- Configure JMS servers and destinations using WebLogic.

Course Outline:**Introduction to WebLogic**

What is WebLogic?
Overview of WebLogic
WebLogic Directory Structure
The config.xml File
Starting and Stopping WebLogic
Administration Console
WebLogic Development Environment Setup

JMS Concepts

Introduction
JMS and the J2EE Platform
Basic JMS Concepts
The JMS Programming Model
Point-to-Point Example – Sender
Point-to-Point Example – Receiver
Configuring JMS for WebLogic
Running the Point-to-Point Example
Publish/Subscribe Example - Publisher
Publish/Subscribe Example – Subscriber
Running the Publish/Subscribe Example

JMS Message Format

Message Header Fields
Message Properties
Message Selectors
Using Other Message Formats

JMS Reliability

Reliable Message Delivery
Message Acknowledgement
Message Persistence
Configuring a JDBC Connection Pool
Configuring Data Sources
Configuring a JMS Store in WebLogic
Viewing the JMS JDBC Store
Temporary Destinations
Enabling Temporary Destinations in WebLogic
Durable Subscriptions
JMS Transactions
JMS Transactions – Example
Special Considerations for JMS Transactions

Appendix A: Java Naming and Directory Interface (JNDI)

What is JNDI?
Benefits of JNDI
Naming Services
Directory Services
Using JNDI
Context Operations
JNDI Utility Class
JNDI Example
Running the JNDI Example

Naming Exceptions

Appendix B: Message-Driven Beans

Message-Driven Beans
Message-Driven Bean Example
Running the Message-Driven Bean Example

Appendix C: WebLogic JMS Extensions

Setting Message Delivery Times
Creating Destinations Dynamically

Course Description:

This 5-day course will introduce Java programmers to the Swing package. The Swing package, extending the AWT, provides efficient and easy-to-use tools for manipulating graphics. Students completing this course will have the ability to create fully-functional, customizable, graphical user interfaces.

Who Should Attend:

This course is designed for Java programmers of any level.

Prerequisites:

Java programming experience is required. Experience with the AWT is helpful, but not required.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Describe the key features of Swing
- Contrast the use of Swing and the AWT
- Add components to a JFrame using the ContentPane
- Write Swing Applications using Swing components such as JLabel, JButton, JTextField, JTree, etc.
- Implement keystroke handling in response to keyboard events from a user
- Understand InputMap and ActionMap
- Use the printing capability to print GUIs
- Describe the Model View Controller Architecture
- Encode Model View Architecture components
- Write code that uses Border objects to alter the appearance of components
- Create programs using various Pane components such as JTabbedPane, JSplitPane, JScrollPane, and JOptionPane
- Write programs that support cut/paste and drag/drop

Course Outline:**An Introduction to Swing**

Rational for the Swing Components
Java Foundation Classes
Lightweight Components
Features of the Swing Components
The AWT Components
An Example Using the AWT Components
Building a Minimum Swing Application

Window Events

Introduction
Window Positioning
Handling Window Events
Window Closing Styles
Inner Classes
Anonymous Inner Classes

Simple Swing Components

Introduction
JButton with Selected Fonts
JButton with an Icon
The Icon Interface
JButton with a Mnemonic
Labels

Keystrokes

The Focus
Focus Traversal
Keyboard Input
Keyboard Shortcuts
Keystrokes and Actions

Dialogs

What is a Dialog?
A Simple Dialog
Adding Components to a Dialog

Menus and Toolbars

AWT vs. Swing Menus
Adding Mnemonics
Adding Accelerators
Adding Colors Fonts, and Icons
Toolbars

Painting and Graphics

Painting Basics
The paintComponent Method
Some Methods of the Graphics Class
Creating a Drawing Area
Drawing Area Source Code

Printing

Printing Basics

A Printing Example
Using the Book Class

Layout Managers

Introduction
FlowLayout
GridLayout
BorderLayout
CardLayout
GridBagLayout
GridBagConstraints
BoxLayout
OverlayLayout

Models, Views, and Controllers

Introduction
The MVC Process of JButton
Advantages to using the MVC Architecture
Pluggable Look and Feel
An MVC Example
Linking the Components
MVC Diagram
ListModel
AbstractListModel
JTable
TableModel
MVCList Source Code

Using the JTree

Tree Basics
The DefaultMutableTreeNode
DefaultTreeModel
Constructing a JTree
Customizing the JTree
Tree Listeners

Text Components

Various Text Components of Swing
JTextField
JPasswordField
JTextArea
Using a JScrollPane
JTextPane
StyleConstants
StyleTest Source Code

Utility Panes, Borders, and Focus

Various Utility Panes
JTabbedPane
JSplitPane
JFileChooser
JOptionPane

Using Borders

Creating Custom Components

Extending JComponent
Creating JButton
JButton's Model to View
The Constructor
More with Listeners
JButton Source Code

Clipboard and Drag & Drop

Clipboard
DataFlavors
System Clipboard vs. Custom Clipboard
More with DataFlavors
Drag & Drop

Appendix A: Events and Their Listeners

AWT Events
Swing Events

Appendix B: Accessibility

Accessible Applications
The ADA and Section 508 Regulations
Accessibility Needs
Java Support for Accessibility
The Java Accessibility API
AccessibleContext
Section 508 Accessibility Standards
Section 1194.21 (a) - Keyboard Access
Section 1194.21 (b) - Accessibility Features
Section 1194.21 (c) - Input Focus
Section 1194.21 (d) - Object Information
Section 1194.21 (e) - Bitmap Images
Section 1194.21 (f) - Textual Information
Section 1194.21 (g) - User Selected Attributes
Section 1194.21 (h) - Animation
Section 1194.21 (i) - Color Coding
Section 1194.21 (j) - Color and Contrast
Section 1194.21 (k) - Flicker Rate
Section 1194.21 (l) - Electronic Forms
The Java Accessibility Bridge
Resources

Course Description:

This course introduces the Spring Framework, the leading full-stack framework for Java EE applications. Topics covered include the Spring container, dependency injection, data validation, aspect-oriented programming, the JDBC Template, and the Hibernate Template. A Web application is also presented to illustrate the use of the Spring Web MVC design pattern.

Who Should Attend:

This course is for Java developers and architects who wish to explore a popular, open-source alternative to traditional Java EE programming.

Prerequisites:

Students must have strong Java programming skills and exposure to Java EE technology.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Describe the seven component modules of the Spring Framework.
- Understand the basic philosophies of Spring.
- Explain the purpose and benefits of dependency injection.
- Configure beans in a Spring configuration file.
- Use setter and constructor injection with Spring beans.
- Create property files for error messages and to support internationalization.
- Write validators and property editors for user-defined data .
- Understand basic concepts of aspect-oriented programming.
- Use the JDBC template to simplify database access.
- Use the Hibernate template to integrate Hibernate and Spring.
- Create Web applications using the Model-View-Controller architecture.
- Write forms and controllers for Spring Web applications.

Course Outline:**Introduction to Spring**

What is Spring?
Overview of the Spring Framework
Spring Philosophies
Spring Documentation
Java 5 Language Features

A First Look at Spring

A Simple Example
Wiring Beans
Configuring a Properties File
Schema-Based Configuration

Beans and Containers

Spring Containers
Spring Configuration File
Spring Beans
Using the Container
The BeanFactory Interface
Singleton vs. Prototype
Bean Naming
Dependency Injection
Setter Injection
Constructor Injection

The Application Context

The ApplicationContext Interface
Accessing Application Components
Accessing Resources
Internationalization with MessageSource
Application Events

Data Validation and Conversion

The Validator Interface
The Errors Interface
The ValidationUtils Class
Validator Example
Testing the Validator
Property Editors
Custom Property Editors

Aspect-Oriented Programming

Aspect-Oriented Programming
AOP Concepts
AOP Proxies
The AOP Alliance
Types of Advice
AOP Example
Introductions

Using JDBC with Spring

A Simpler Approach
Working with the HSQLDB Database
The JdbcTemplate Class
Exception Translation
Updating with the JdbcTemplate
Queries using the JdbcTemplate
Mapping Results to Java Objects

Using Hibernate with Spring

What is Hibernate?
Hibernate Sessions
The HibernateTemplate
Sample Class and Mapping File
Creating and Saving a New Entity
Locating an Existing Entity
Updating an Existing Entity
Hibernate Query Language
Executing Queries

Spring Web MVC - Part 1

Spring Web MVC
The DispatcherServlet
Writing a Controller
A Simple View Page
Configuring the Controller
Adding a View Resolver
Adding a Message Bundle
Adding Business Classes
Adding Test Data
Accessing a Database
Adding a Form
Updating the Database
Integrating Hibernate

Spring Web MVC - Part 2

Handler Mappings
View Resolution
Chaining View Resolvers
Controllers
AbstractWizardFormController

Appendix A: Resources

Resources

Appendix B: Spring IDE

Spring IDE
Adding the Spring Project Nature
Managing Spring Configuration Files
Visualizing a Configuration File

Appendix C: Creating a Spring Project in Eclipse

Introduction
Creating a Spring Project
Configuring the Build Path

Appendix D: Running the Examples in Eclipse

Running the Standalone Applications
Starting the Server
Starting the Server
Running the Web Application
Modifying the Web Application
Stopping the Server

Course Description:

This course introduces Hibernate, a popular open-source object/relational mapping (ORM) tool that helps Java developers store and access persistent objects. Topics covered include Hibernate configuration, the Hibernate mapping file, inheritance, collections, associations, and the Hibernate Query Language (HQL).

Who Should Attend:

This course is for Java developers creating or maintaining applications that use a relational database and Java SE or Java EE, Java developers and architects investigating ORM alternatives.

Prerequisites:

Students should have Java programming experience and knowledge of Structured Query Language (SQL).

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Describe the purpose and benefits of an object/relational mapping tool
- Configure database connection properties in the Hibernate configuration file
- Use a Hibernate mapping file to map a Java class to a database table
- Create, save, update, and delete entities
- Distinguish between entity and value types
- Configure primary key generators for persistent classes
- Describe and use the Hibernate strategies for mapping inheritance hierarchies
- Map collections and associations
- Write queries using Hibernate Query Language (HQL)

Course Outline:**Getting Started with Hibernate**

What is Hibernate?
Using Hibernate
Configuring Hibernate
Hibernate Sessions
Writing Classes for Hibernate Applications
Sample Class and Mapping File
Creating and Saving a New Entity
Locating an Existing Entity
Updating an Existing Entity
Deleting an Entity
Executing an SQL Query
Programmatic Configuration

Mapping Persistent Classes

The Hibernate Mapping File
Entities and Values
Class Mappings
Properties
Derived and Generated Properties
Mapping Value Types
Key Generators
Compound Keys
Hibernate Types

Inheritance

Mapping Class Inheritance
Table Per Class Hierarchy
Table Per Subclass
Table Per Concrete Class
Using Implicit Polymorphism

Collections and Associations

Mapping Collections
Collections of Components
Sample Application - UML Diagram
Sample Application - Database Schema
Implementing Associations
Mapping Associations
The `inverse` Attribute

Hibernate Query Language

HQL Basics
HQL Expressions
HQL Functions
Polymorphic Queries
Executing Queries
Scrollable Results
Named Queries
Associations and Joins
Inner Joins
Outer Joins
Sample Queries

Appendix A: XDoclet and Java Annotations

Using XDoclet Markup
Using Annotations

Course Description:

This course shows JSP and Servlet developers how to build web applications using the Apache Struts framework. Students learn to use the Struts architecture to develop web applications using the Model-View-Controller (MVC) design pattern and Struts custom tags to create JSPs and Servlets that adhere to an industry standard.

Who Should Attend:

This course is for Java 2 Platform, Enterprise Edition (J2EE) application developers.

Prerequisites:

Students should be familiar with Java Servlets and JavaServer Pages (JSPs), and have experience with Web application development.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Describe the Struts 2 Architecture
- Describe the Model-View-Controller (MVC) Architecture
- Create a basic Struts 2 Web application
- Create and modify Struts configuration files
- Create action classes from POJOs
- Create action classes by extending existing framework
- Work with results to present views to the client
- Understand and apply exception mappings
- Implement interceptors
- Use interceptors to assist in programmatic data validation
- Utilize the the OGNL expression language
- Understand the Value Stack used by OGNL
- Implement data tags within the client view
- Implement generic tags within the client view
- Utilize form tags to tie the actions to the JSP views

Course Outline:**The Evolution of a Web Application**

What Is a Web Application?
A Static HTML Resource
HTML Forms
Servlets
JavaServer Pages (JSP)
JavaBeans, Custom Tags, and JSTL
Web Application Frameworks

Overview of the Struts 2 Framework

Tiered Architectures
Model-View-Controller (MVC) Architecture
The Struts 2 Framework
Bundling the Struts 2 Libraries
Creating a Basic Struts 2 Web Application

Struts Configuration Files

An Overview of Struts Configuration Files
The struts.xml Configuration File
The package Element
The action Element
The result Element
The struts.properties Configuration File
A HelloStruts Application
Examining the HelloStruts Application

Actions in Detail

POJOs as Actions
Implementing the Action Interface
Actions Accessing Resources
A
The ActionSupport Class

Results and Result Types

The result Element
Standard Result Types
Global Results and Exception Mappings
SearchingForResults Application
A ButtonBuilder Application

Interceptors

An Overview of Interceptors
Struts Predefined Interceptors
A Simple Example
Interceptor Details
A Progress Meter Example
Programmatic Validation of Data

OGNL and the Value Stack

An Overview of OGNL
An Overview of the Value Stack
OGNL with Java Collections and Maps

OGNL With Collections and Maps
Additional OGNL Features

Generic Tags

Generic Data Tags
Generic Control Tags
Data and Control Tags Reference

Form Tags

Form Tags
Shared Form Tag Attributes
Form Tags Reference

Appendix A: Software Libraries Reference Sheet

Installed Software Directories

Appendix B: Developing Java Web Applications With Eclipse

What is Eclipse™?
Starting Eclipse
Configuring the Workspace
Configuring a Server in Eclipse
Configuring the Apache Tomcat Server
Installing the GlassFish Plug-in for Eclipse
Configuring the GlassFish Application Server
Configuring the JBoss Application Server
Configuring Server Publishing
Starting and Stopping Servers in Eclipse
Configuring Apache Derby in Eclipse
Configuring MySQL in Eclipse
Creating a Dynamic Web Project
Creating an HTML Page
Creating a JSP Page
Creating a Servlet
Configuring the Build Path

Appendix C: HTML Reference

Introduction
A Simple HTML Document
Basic Tags
Formatting Tags
Links
Forms

Appendix D: Web Accessibility

What is Accessibility?
What is Section 508?
Accessibility Initiatives and Related Legislation
Types of Disabilities
Assistive Technologies
Benefits of Accessible Design
General Coding Practices
Other Recommendations

Course Description:

This course is designed to implement a Tomcat Servlet Container. The course covers the installation and configuration of Tomcat as a secure corporate Servlet Container capable of virtual hosting and linking to the Apache Web Server. Advanced features of Valves, Connectors, the JNDI name service, and JDBC connection pooling are also discussed.

Who Should Attend:

The course is designed for Systems Administrators that wish to implement Tomcat in a production environment. The class is taught with Tomcat running on the UNIX operating system, therefore students should be comfortable with basic UNIX commands.

Prerequisites:

Students should have familiarity with the UNIX operating system is assumed. Experience with Java Web applications is a plus.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Install, monitor, and maintain a secure corporate Tomcat Servlet Container.

Course Outline:**Overview of the Tomcat Project**

What is it?
History
Tomcat version vs. Servlet/JSP spec. versions
Logical outline of the Tomcat components

Installation and Basic Configuration

Downloading and installing manually (UNIX/Linux and Windows)
Installing automatically (Windows)
Running Tomcat
Running Tomcat as a daemon or Windows service
Tomcat directories
Basic settings in the server.xml file (Ports for HTTP and shutdown)
Adding users to the basic Tomcat security

Web Application Overview

What is a Web Application?
Static Files (HTML, GIF, JPG, etc.)
Servlets
JSPs
Basic configuration (web.xml)
WAR files
Installing WAR files into Tomcat

Advanced Tomcat Configuration

Valves
Connectors
JNDI Configuration
JDBC Connection Pool Configuration

Classloaders

What are they?
Why do we care?
Common classloader problems and solutions

Security

System-Level Security
Securing Web Applications

Shared Hosting

What is it and why?
Creating separate domains for each host
Creating separate CATALINA_HOMES for each host
Configuring each host with a separate JVM.

Tomcat with Apache

What is Apache?
Why use Apache with Tomcat?
Installing Apache

Load Balancing

Sticky vs. non-sticky sessions
Using the stupid Balancer.war application (bad bad bad)
Using Apache to load balance multiple Tomcats

Course Description:

The JavaServer Faces framework is establishing itself as the new standard for the development of web applications. Designed under Sun's Java Community Process by many of the same people that developed the Jakarta Struts framework, Faces is proving itself to be the next step in the web application evolutionary process. JavaServer Faces, or simply JSF, combines Java Servlets and JavaServer Pages into a server-side implementation of the Model-View-Controller Design Pattern. The JSF framework provides developers with a unified infrastructure upon which Internet applications can be constructed.

The course also introduces the developer to the JSF architecture and provides the basis for planning, developing, and deploying Web based applications using the JSF framework. After taking this class, the developer will be able to quickly construct dynamic server-side web pages using JSF and integrate the Web application with many of the other Java2 Enterprise Edition application server methodologies such as Enterprise Java Beans, JavaMail, and SOAP. This class combines lecture with a unifying, hands-on experience, and open discussion that will help the developer quickly understand the benefits of JSF and how to use the framework.

Who Should Attend:

This course is for Java Developers.

Prerequisites:

Participants should already have a solid understanding of Java programming and understand the basics of XML. The course also assumes a basic understanding of HTML syntax and JavaServer Pages syntax. Understanding of Enterprise Java Beans (EJB) is also a plus.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Write web applications that take advantage of the FacesServlet, FacesContext and Action Java classes to control the user experience of the web application.
- Write JSF applications that gather and update information from external application servers such as EJBs, CORBA servers, and database servers.
- Create and use custom Tag Libraries in JavaServer Pages.
- Understand the basics of web security and learn to take advantage of the security features provided by the Web Server.
- Understand the use of the standard JSF Validators and how to write custom Validators.
- Understand the use of the standard JSF Data Conversion classes and how to write custom Data Converters.
- Take advantage of the JSF architecture that supports rendering output in several formats from the same application. Such as: HTML, WML, XML, etc.

Course Outline:

Introduction and Overview

The JSF Architecture

JSF Request Objects

Simple JSF User Interface components

The EL Expression Language and Advanced User Interface components

Event Handling

Data Validation

Data Conversion

Rendering Custom User Interface Objects

Course Description:

This course introduces the latest version of Maven, a popular open-source build automation and software comprehension tool. Core Maven concepts are demonstrated in a series of hands-on lab exercises.

Who Should Attend:

This course is designed for software developers, configuration management specialists, and project managers with no previous experience using Maven.

Prerequisites:

Students should have some familiarity with the Java programming language.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Describe the objectives and benefits of Maven
- Create, locate, and utilize Maven projects by using Maven project coordinates
- Create projects from Maven archetypes
- Use and distinguish between local and remote Maven repositories
- Understand the Maven Project Object Model (POM)
- Create and manage multi-module projects
- Use features of Maven dependency management
- Generate a project Web site

Course Outline:**Getting Started with Maven 3**

What is Maven?
Maven's Objectives
Comparing Maven and Ant
Installing Maven
Installation Details
Compatibility with Maven 2
Updating Maven

Maven Core Concepts

Maven Repositories
Project Coordinates
Version Identifiers
Standard Project Directory Structure
Plugins and Goals
Maven Default Lifecycle
The `target` Directory
Lifecycle Bindings
The Clean Lifecycle
Archetypes

The Project Object Model (POM)

The `pom.xml` File
Describing Your Project
Customizing Plugin Behavior
Property References
Multi-Module Projects
Project Inheritance
The Super POM
The Effective POM

Dependency Management

Maven's Dependency Mechanism
Dependency Version Ranges
Dependency Scope
System Dependencies
Viewing Dependencies
Transitive Dependencies
Optional Dependencies
Excluding Dependencies
Inheriting Dependencies

Customizing Maven Builds

The `settings.xml` File
Mirrors
Servers
Proxies
Plugin Groups
Properties
User-Defined Properties
Resource Filtering

Site Generation and Reports

The Site Lifecycle
Site Directory Structure
Almost Plain Text (APT)
FAQ Markup Language (FML)
Customizing the Site Descriptor

Default Reports
Other Reporting Plugins

Appendix A: Reference

Resources
Maven Command Line Options
Integration with Eclipse
Testing with JUnit
Maven Plugin for C/C++ Development

Course Description:

MVN-201 Designing Development Infrastructure covers topics for the advanced Maven user such as advanced multimodule project architecture, enforcing standards with the Enforcer plugin, installing and configuring a repository manager, and installing and configuring a continuous integration server

Who Should Attend:

This class is designed for Development Infrastructure Engineers.

Prerequisites:

Students should have familiarity with the content covered in Maven Mechanics.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Understand the structure of a Maven POM and a Maven Multi-module Project.

Course Outline:**Module 1: Consistent Builds**

Standardized Environment
Explicit Use of Plugin Versions
The Enforcer Plugin

Module 2: Site Generation with Maven

Maven Site as Collaboration Tool
The Site Lifecycle
Default Site Layout
Creating a Site Project
Organizing Your Site
Adding Site Content

Module 3: Site Reporting with Maven

Technical Documentation
Browsable Source Code
Displaying Test Results
Code Quality Metrics
Lab 1: Adding Reports to Your Site

Module 4: Web Development

Using the Maven Jetty Plugin
Developing Web Applications with Maven
Lab 2: Running Your Webapp in Jetty

Module 5: Repository Management

What is an Enterprise Repository?
Installing Nexus
Using Nexus
Nexus Repositories and Security
Deploying to Nexus with Maven
Managing Maven Settings with Nexus
Enterprise Security (LDAP)
Configuring Procurement in Nexus
Performing a Staged Release

Module 6: Release Management

Release Management
The Maven Release Plugin
Developer Release Workflow
Integration with Source Control
Example Using Subversion
Example Using Git
Lab 3: Using Nexus

Module 7: Continuous Integration with Hudson

What is Continuous Integration?
Benefits of Continuous Integration
Downloading Hudson
Installing Hudson
Running Hudson
Configuring Hudson for Maven
Setting Up a Maven Job in Hudson
Monitoring Hudson Builds
Demonstration of Hudson
Lab 4: Using Hudson

Course Description:

This course is a comprehensive introduction to writing Enterprise JavaBeans (EJB) using the EJB 3.x specification. An overview of EJB3 is provided, followed by hands-on experience with session beans, entity beans, and message-driven beans. Topics also include container-managed relationships (CMR) and the EJB Timer Service. This course can be taught using IBM WebSphere, Oracle WebLogic, or the JBoss application server.

Who Should Attend:

This course is for Java programmers and software engineers with experience writing enterprise JavaBeans and who wish to update from EJB2 to EJB3.

Prerequisites:

Students should be comfortable with Java programming and object-oriented concepts. In addition, students should have prior experience developing EJBs according to the EJB 2.x specification.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Start, stop, and configure an application server
- Use JNDI to access database and EJB resources
- Create JDBC data sources
- Write stateless and stateful session beans
- Use container-managed persistence
- Use standard annotations for EJBs
- Configure and deploy EJB applications
- Assemble Java EE-compliant enterprise applications
- Implement container-managed relationships
- Write message-driven beans
- Use the EJB Timer Service

Course Outline:**Java EE Development using Eclipse**

What is Eclipse?
Starting Eclipse
Creating a Java Project
Importing Existing Java Code
Creating a New Java Program
Java EE-Compliant Application Servers
Creating a Server Instance
Starting and Stopping the Server

Using JDBC Data Sources

A Simple JDBC Program
JDBC Driver Types
Using the Derby Database
JDBC Data Sources
Data Source Example
Configuring a JDBC Data Source
Running the JDBC Examples
Executing a Query
Using the Database Explorer

Introduction to EJB 3

Limitations of EJB 2
EJB 3 Feature Overview
Comparing EJB 2 and 3
The EJB 3 Business Interface
Review of Java Annotations
The Annotated EJB Class
Dependency Injection
Container Callback Methods

Session Beans

Session Bean Lifetime
Session Bean Interface
Session Bean Lifecycles
Stateless Session Bean Example
Accessing Environment Entries
Stateful Session Bean Example
EJB Exceptions - Examples
Testing the Session Beans
Creating a New Session Bean

CMP Entity Beans

Entity Beans
Entity Bean Interface
Lifecycle of an Entity Bean
Container-Managed Persistence
Primary Key Class
Implementing CMP Entity Bean Methods
Container-Managed Persistence Example
Custom Finders
EJB Query Language
Mapping Container-Managed Fields

Container-Managed Relationships

Container-Managed Relationships
Local Interfaces
Local Home Interfaces
CMR Example - Entity Bean Classes
Transfer Object Pattern
CMR Example - Session Bean
Creating New CMP Entity Beans
Creating a Relationship

Message-Driven Beans

Message-Driven Beans
Message-Driven Bean Lifecycle
Message-Driven Bean Example
Deploying Message-Driven Beans
Creating a New Message-Driven Bean

EJB Timer Service

Overview of the Timer Service
Timer Service API
Creating Timers
Canceling and Saving Timers

Appendix A: Web Resources

Java Technology
Application Servers
Derby Database

Course Description:

This course is a comprehensive introduction to writing Enterprise JavaBeans (EJB) using the EJB 3.x specification. An overview of Java EE technology is provided, followed by hands-on experience with JNDI, JDBC, JMS, session beans, entity beans, and message-driven beans. Topics also include container-managed persistence (CMP), container-managed relationships (CMR), and the EJB Timer Service. This course can be taught using IBM WebSphere, Oracle WebLogic, or the JBoss application server.

Who Should Attend:

This course is for experienced Java programmers and software engineers preparing to write Enterprise JavaBeans for Java EE applications hosted on a Java EE-compliant application server.

Prerequisites:

Students should be comfortable with Java programming and object-oriented concepts. A minimum of six months coding experience is suggested. In addition, students should have prior experience using JDBC and SQL.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Start, stop, and configure an application server
- Use JNDI to access database and EJB resources
- Create JDBC data sources
- Write stateless and stateful session beans
- Use container-managed persistence
- Use standard annotations for EJBs
- Configure and deploy EJB applications
- Assemble Java EE-compliant enterprise applications
- Use the Session Facade pattern
- Implement container-managed relationships
- Create JMS destinations
- Write message-driven beans
- Use the EJB Timer Service

Course Outline:**Overview of Java EE**

Java Platforms
Characteristics of "Enterprise" Computing
Java EE Technologies
Multi-Tier Architectures
Advantages of Multi-Tier Architectures
Container-Based Approach
Parties Involved in Java EE Deployment
Java EE Application Models
HTTP Services Application Model
N-Tiered Application Model

Java EE Development using Eclipse

What is Eclipse?
Starting Eclipse
Creating a Java Project
Importing Existing Java Code
Creating a New Java Program
Java EE-Compliant Application Servers
Creating a Server Instance
Starting and Stopping the Server

Java Naming and Directory Interface

What is JNDI?
Benefits of JNDI
Naming Services
Directory Services
Using JNDI
Context Operations
JNDI Utility Class
JNDI Example
Naming Exceptions
Running the JNDI Example

Using JDBC Data Sources

A Simple JDBC Program
JDBC Driver Types
Using the Derby Database
JDBC Data Sources
Data Source Example
Configuring a JDBC Data Source
Running the JDBC Examples
Executing a Query
Using the Database Explorer

RMI and IOP

Object Serialization
Remote Method Invocation

RMI Architecture
The Remote Interface
CORBA

Enterprise JavaBeans

Enterprise JavaBeans Component Model
Types of Enterprise Beans
EJB Wrapper Interfaces
Deployment Descriptors
Context and Environment Objects
EJB Runtime Environment
The Remote Interface
The Home Interface
The Enterprise Bean Class
The Client Test Program
The ejb-jar.xml File
The Vendor-Specific Deployment Descriptor
Creating an Enterprise Application Project
Deploying the Enterprise Application

Introduction to EJB 3

Limitations of EJB 2
EJB 3 Feature Overview
Comparing EJB 2 and 3
The EJB 3 Business Interface
Review of Java Annotations
The Annotated EJB Class
Dependency Injection
Container Callback Methods

Session Beans

Session Bean Lifetime
Session Bean Interface
Session Bean Lifecycles
Stateless Session Bean Example
Accessing Environment Entries
Stateful Session Bean Example
EJB Exceptions - Examples
Testing the Session Beans
Creating a New Session Bean

CMR Entity Beans

Entity Beans
Entity Bean Interface
Lifecycle of an Entity Bean
Container-Managed Persistence
Primary Key Class
Implementing CMP Entity Bean Methods
Container-Managed Persistence Example
Custom Finders

EJB Query Language
Mapping Container-Managed Fields

Session Facade Pattern

J2EE Design Patterns
Session Facade Pattern
Local Interfaces
Example - ItemOrderer Bean
Deployment Settings for ItemOrderer Bean
Testing the Session Bean

Container-Managed Relationships

Container-Managed Relationships
Local Interfaces
Local Home Interfaces
CMR Example - Entity Bean Classes
Transfer Object Pattern
CMR Example - Session Bean
Creating New CMP Entity Beans
Creating a Relationship

Message-Driven Beans

Message-Driven Beans
Message-Driven Bean Lifecycle
Message-Driven Bean Example
Deploying Message-Driven Beans
Creating a New Message-Driven Bean

EJB Timer Service

Overview of the Timer Service
Timer Service API
Creating Timers
Canceling and Saving Timers

Appendix A: Web Resources

Java Technology
Application Servers
Derby Database

Appendix B: Using EJBs in a Web Application

Using Web Components as EJB Clients
Servlet Code for the Survey Application
Session Bean for the Survey Application
Deploying the Survey Application

Appendix C: EJB Transactions

Transactions
Container-Managed Transactions

Transaction Attributes
System vs. Application Exceptions
Rolling Back a Container-Managed Transaction

Appendix D: EJB Security

Java EE Security
Specifying Permissions for EJBs

Appendix E: Java Message Service

Introduction
JMS and the Java EE Platform
Basic JMS Concepts
The JMS Programming Model
Point-to-Point Example
Configuring JMS
Publish/Subscribe Example
Reliable Message Delivery

Course Description:

This course introduces the student to the Python language. Upon completion of this class, the student will be able to write non trivial Python programs dealing with a wide variety of subject matter domains. Topics include language components, the IDLE environment, control flow constructs, strings, I/O, collections, classes, modules, and regular expressions. The course is supplemented with many hands on labs using either Linux or Windows.

Who Should Attend:

This course is designed for anyone who needs to learn how to write programs in Python.

Prerequisites:

Students should have taken the Software Development for Non-Programmers course or have some experience with at least one programming language. Typically, students in this course will have already programmed in either C, C++, Java, Perl, Ruby, VB, or anything equivalent to these languages.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Execute Python code in a variety of environments
- Use correct Python syntax in Python programs
- Use the correct Python control flow construct
- Write Python programs using various collection data types
- Write home grown Python functions
- Use many of the standard Python modules such as os, sys, math, and time
- Trap various errors via the Python Exception Handling model
- Use the IO model in Python to read and write disk files
- Create their own classes and use existing Python classes
- Understand and use the Object Oriented paradigm in Python programs
- Use the Python Regular Expression capabilities for data verification

Course Outline:**An Introduction to Python**

Introductory Remarks about Python
Strengths and Weaknesses
A Brief History of Python
Python Versions
Installing Python
Environment Variables
Executing Python from the Command Line
IDLE
Editing Python Files
Getting Help
Dynamic Types
Python Reserved Words
Naming Conventions

Basic Python Syntax

Introduction
Basic Syntax
Comments
String Values
String Operations
The `format` Method
String Slices
String Operators
Numeric Data Types
Conversions
Simple Input and Output
The `print` Function

Language Components

Introduction
Control Flow and Syntax
Indenting
The `if` Statement
Relational Operators
Logical Operators
True or False
Bit Wise Operators
The `while` Loop
`break` and `continue`
The `for` Loop

Collections

Introduction
Lists
Tuples
Sets
Dictionaries
Sorting Dictionaries
Copying Collections
Summary

Functions

Introduction
Defining Your Own Functions
Parameters
Function Documentation
Keyword and Optional Parameters
Passing Collections to a Function
Variable Number of Arguments
Scope
Functions - "First Class Citizens"
Passing Functions to a Function
Mapping Functions in a Dictionary
Lambda
Closures

Modules

Modules
Standard Modules - `sys`
Standard Modules - `math`
Standard Modules - `time`
The `dir` Function

Exceptions

Errors
Run Time Errors
The Exception Model
Exception Hierarchy
Handling Multiple Exceptions
`raise`
`assert`
Writing Your Own Exception Classes

Input and Output

Introduction
Data Streams
Creating Your Own Data Streams
Access Modes
Writing Data to a File
Reading Data From a File
Additional File Methods
Using Pipes as Data Streams
Handling IO Exceptions
Working with Directories
Metadata
The `pickle` Module

Classes in Python

Classes in Python
Principles of Object Orientation
Creating Classes

Instance Methods
File Organization
Special Methods
Class Variables
Inheritance
Polymorphism
Type Identification
Custom Exception Classes
Class Documentation - `pydoc`

Regular Expressions

Introduction
Simple Character Matches
Special Characters
Character Classes
Quantifiers
The Dot Character
Greedy Matches
Grouping
Matching at Beginning or End
Match Objects
Substituting
Splitting a String
Compiling Regular Expressions
Flags

Course Description:

The course covers a handful of various Python advanced topics including high level data structures, network programming, writing GUI's in Python, and CGI programming.

Who Should Attend:

This course is for students wanting to further their knowledge of Python.

Prerequisites:

Students should have taken an introductory Python course or have six months of Python programming experience.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Write Network Programs in Python
- Write CGI programs in Python
- Write GUI programs in Python
- Differentiate between the proper use of Python collection types
- Use advanced Data Structures
- Become proficient in the use of bit variables
- Use Python List comprehensions
- Use Python generators
- Use the most popular Python modules
- Create and execute processes

Course Outline:**What You Should Already Know About Python**

Introduction
 Language Evolution
 Python Reserved Words and Other Rules
 Documentation
 The `string` Class
 Variables
 Data Types
 Boolean and Numeric Types
 Strings
 Lists and Tuples
 Sets
 Sequences
 Looping Through Sequences
 Dictionaries
 Bit Variables
 Modules
 Reading Files
 Some File Tests

Data Structures

`range`
 List Comprehensions
 Nested List Comprehensions
 Dictionary Comprehensions
 Dictionaries with Compound Values
 Processing Lists in Parallel
 Functions
 Default Parameters
 Variable Arguments
 A Dictionary of Dictionaries
 Specialized Sorts
 The `del` Statement
 Time Functionality
 Using Generators

Writing GUIs in Python

Introduction
 Components and Events
 An Example GUI
 The `root` Component
 Adding a Button
 Entry Widgets
 Text Widgets
 Checkbuttons
 Radiobuttons
 Listboxes
 Frames
 Menus
 Binding Events to Widgets

Python and CGI Scripts

Introduction
 HTML
 HTML Forms
 A Guestbook Application

What Can Go Wrong!
 HTML Tables
 The CGI Script
 Rendering of the Script

The `os` Module

The Environment
 Launching Commands
 Creating a Process
 Directory Commands
 Other Process Methods
 File Information (Metadata)
 Miscellaneous `os` Calls
 Walking Through Directories

Network Programming

Introduction
 A Daytime Server
 Clients and Servers
 The Client Program
 The Server Program
 Recap
 An Evaluation Client and Server
 The Server Portion
 A Threaded Server

Course Description:

Perl is a scripting language which allows for rapid prototyping of projects formerly done with a programming language or a shell. It incorporates all the functionality of C (including a UNIX system interface), the Shells, grep, sed, and awk. The topics in the course will aid all computer users - from end user to programmer to administrator alike. Many in-class labs support the course material.

Who Should Attend:

This course is for programmers, end users, system administrators, network administrators, CGI script writers, or anybody who is interested in automating tasks but doesn't want to learn all the details of a full blown programming language.

Prerequisites:

Students should have some experience with either a programming language (preferably C), or any of the UNIX shells.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Manipulate files and directories
- Use arrays and array functions to solve a wide variety of problems
- Use the powerful regular expression capabilities of Perl
- Generate reports
- Use hashes to solve commonly encountered problems
- Take advantage of Perl's powerful system interface
- Write programs that solve many system administrator problems
- Use Perl to write CGI applications
- Use Modules from the standard Perl distribution
- Use Perl references

Course Outline:**Getting Started with Perl**

What is Perl?
Where Can I Get Perl?
A Simple Perl Program
Simple I/O
Perl Variables
Control Flow - Decisions
Control Flow - Loops
Altering Loop Control Flow
Statement Modifiers
What Is True And What Is False?
The Special Variable \$_
The Special Variable \$~

Perl Operators

Introduction
Table Of Perl Operators
Arithmetic Operators
String Operators
Relational Operators
Logical Operators
Bitwise Operators
Assignment Operators
The Conditional Operator
Range Operator
String Functions
The eval Function

I/O

Introduction
String Literals
The print Function
Here Documents
The printf Function
The sprintf Function
Filehandles
Opening Disk Files
File Open Errors
The die and warn Functions
File Operators

Arrays

Basic Concepts
Assigning Values To An Array
Accessing Array Elements
Array Functions
push and pop
shift
sort, reverse, and chop
split and join
grep
splice
Command Line Arguments

Associative Arrays

Basic Concepts
Associative Array Functions
Updating Associative Arrays
Accessing Environment Variables

Subroutines

Calling Subroutines
Passing Arguments to Subroutines
Returning Values from Subroutines
The require Function
Packages and Modules
The @INC Array
Predefined Subroutines
Comparison Subroutines for Sorting

Pattern Matching and Regular Expressions

Introduction
Regular Expression Syntax
The Match Operator
Regular Expression Meta-Characters
Anchors
Single Character Matches
Some Special Issues
Character Classes
Multiple Character Matches
Alternation
The Substitution Operator
The Translation Operator

Accessing System Resources

Introduction
File and Directory System Calls
The stat Function
The utime Function
The fork Function
The exec and wait Functions
Handling Signals
The system Function
Command Substitution
Opening Pipe Files

Generating Reports with Perl

Formats
Formatting Examples
Multi-Line Values
Multi-Line Text Blocks
Sending a Report to a File
The select Function
The Special Variable \$~

Top-of-Page Formats
Bottom-of-Page Formats
A Sample Report

Perl and CGI

What is CGI?
Web Servers and Browsers
HTML
HTML Forms
Form Elements
A Typical CGI Application
CGI Input
CGI Output
Using the CGI.pm Module
CGI Environment Variables

Appendix A: Command Line Options and Debugging

Running Perl on the Command Line
Summary of Command Line Options
The Perl Debugger
Perl Debugger Commands
Other Debugging Aids
The strict Module

Appendix B: More About Regular Expressions

Remembered Matches
Greedy Regular Expressions
Nested Remembered Patterns
Matching for Multiple Occurrences in a Loop

Appendix C: References and Data Structures

References
Syntactic Sugar
Anonymous Arrays
Higher Dimensional Arrays
References and Subroutines

Appendix D: Comparing Perl 4 and 5

Operators
Packages
Typeglobs

Appendix E: Advanced Perl

What is in the Advanced Perl Course
A Taste of Object Orientation
A Taste of Network Programming

Appendix F: Perl Development Using Eclipse

Features of the Perl Plugin
Creating a Perl Project
Running a Perl Program
Preferences - EPIC
Preferences - Editor
Preferences - Content Assist
Preferences - Folding
Preferences - Mark Occurrences
Preferences - Templates
Templates
Preferences - Source Formatter
Preferences - Task Tags
Accessing Perl Documentation
Project Properties
Debugging
Testing Regular Expressions

Appendix G: Reference

Special Perl Variables
Regular Expression Meta-Characters
Internet Resources

Course Description:

The course begins with a thorough treatment of packages, modules, and libraries. Next, Perl references are studied. This gives students the necessary background to write object-oriented Perl. Various applications and areas that use object orientation are studied next. These modules include the Tk.pm module for building Graphical User Interfaces, the DBI.pm module, which provides a portable way of querying databases, the CGI.pm module for writing CGI programs, and the Socket.pm module used in client server networking applications. Finally a treatment of XML and Perl is undertaken.

Who Should Attend:

Programmers, end users, system administrators, network administrators, and CGI script writers should attend this course.

Prerequisites:

Participants should be well-versed in the fundamentals of Perl.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Download, install, and use modules from the Comprehensive Perl Archive Network (CPAN).
- Use the modules in the Standard Perl Distribution.
- Write POD (Plain Old Documentation) sections of Perl modules.
- Use Perl references to solve many programming problems including those problems involving arbitrarily complex data structures.
- Distinguish among packages, modules, libraries, and classes and use each one effectively.
- Write client/server applications using the Socket.pm module.
- Write Graphical User Interfaces (GUIs) using the Tk.pm module.
- Write Perl CGI (Common Gateway Interface) scripts.
- Write Perl applications that make queries to real databases through the use of the DBI.pm module.
- Write Perl applications that produce and process XML documents.

Course Outline:**What You Should Already Know**

Introduction
A Quick Review of Perl
Perl Libraries
The Standard Perl Library
Packages
Modules
Using .pm Modules
Exporter.pm
Standard Perl Modules
Comprehensive Perl Archive Network (CPAN)
Roman.pm
Miscellaneous Perl Topics - wantarray

Associative Arrays

Introduction
Associative Arrays as Dual Arrays
A Hashing Algorithm
Collisions
Associative Arrays
Sorting by Keys or Values
Finding Unique Tokens in a File
Reverse Lookups
Selecting the Top n Elements

References

Introduction
Summary of References
Array References
Anonymous Arrays
Anonymous Hashes
Prototypes
Higher Dimensional Arrays
Complex Hashes
References and Subroutines
Anonymous Subroutines
Lists of References

Object-Oriented Programming

Introduction
Object-Oriented Vocabulary
The class Definition
Defining and Using Objects
Information Hiding
Instance Methods

Destructors
Class Methods
Inheritance
Polymorphism
Documenting Perl Code
IO.pm

The TK.PM Module

Introduction
Event Driven Programming
Geometry Management
pack()
grid()
grid() Options
place(): Absolute Coordinates
place(): Relative Coordinates
The Label Widget
The Button Widget
The Checkbutton Widget
The Radiobutton Widget
The Dialog Widget
Text Input Widgets
The Listbox Widget
Menus
Frames
Toplevel Widgets
Bind

Client-Server Applications and CGI

Introduction
Internet Terminology
Data Delivery
Writing a Simple Client
Writing a Simple Server
Writing an Iterative Server
ftp
The Common Gateway Interface
HTML Forms
The CGI Environment
Administering the Server
The HTTP Protocol
Header Information
The CGI Script
Extracting Form Data
The CGI Response

CGI Output
Database Access
What Can Go Wrong?
Images
Extra Path Information

CGI.pm

Using CGI.pm?
Simple Form Elements
Parameters
HTML Tags
Form Processing
checkbox_group and radio_group
Text Areas
Popup Menus and Scrolling Lists
Debugging

Accessing Real Databases in Perl

Introduction
Architecture
Review of SQL
Accessing Databases from Perl
Executing a Query in Perl
Accessing Database Metadata
Interactive Requests
Adding a Graphical Front-End
Accessing a Real Database via a Web Form

XML Fundamentals

Introduction
What is a Markup Language?
SGML vs. HTML
Sample HTML Document
XML
Creating Semantic Tags
XML Syntax
Elements
Attributes
Comments
Unicode and Character Sets
Character References
Entity References
Character Data Sections (CDATA)
Processing Instructions
Parsing XML

Processing XML With Perl

Creating an XML Document With Perl
Creating an XML Document
Using an XML Parser
XML::Simple
XML::Parser

Appendix A: An HTML (P)review

An HTML (P)review
An HTML (P)review: Basics
An HTML (P)review: Formatting Basics
An HTML (P)review: Links
An HTML (P)review: Forms

Course Description:

This course covers the fundamental components of the Ruby Programming Language. Emphasis is placed on the object oriented aspects of Ruby. Topics include arrays, hashes, regular expressions, io, exceptions, modules, and applications areas.

Who Should Attend:

This course is intended primarily for those who have programmed in other programming languages such as, but not limited to, C, C++, Java, or Perl.

Prerequisites:

Students should have taken the Software Development for Non-Programmers course or have at least six months of programming experience in at least one programming language.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Distinguish and use various Ruby datatypes
- Master the use of arrays and hashes
- Build home grown classes
- Use the extensive pre bundled classes
- Use the I/O facilities of Ruby to read and write binary and text files
- Master the use of Iterators to loop through various data structures
- Use Exceptions in handling various run time errors
- Create Ruby modules
- Use the wide variety of Ruby Modules that come with the Ruby distribution

Course Outline:**An Overview Of Ruby (Part 1)**

What is Ruby?
 Executing Ruby Code
 Getting Input
 Comments
 Numbers
 Strings
 The `Class` Class
 Decision Making
 The `case` Construct
 Loops
 Iterators
 Regular Expressions
 Functions
 Odds and Ends
 Time Methods

An Overview of Ruby (Part 2)

Arrays
 Array Operator Methods
 Array Equality Operator
 Arrays as Stacks and Queues
 Higher Dimensional Arrays
 Other Useful Arrays Methods
 Command Line Arguments
 Hashes
 Common Hash Methods
 Sorting Hashes
 Iterators with Arrays and Hashes
 Arrays and Functions
 Hashes and Functions
 Named Parameters
 Symbols
 Procs
 Closures

Classes

Objects
 Brief History of OOP
 OOP Vocabulary
 Creating a New Class
 Using Objects
 Defining Operator Methods
 Inheritance
 Ancestors
`self`
 Access Levels - `public`
 Access Levels - `private`
 Access Levels - `protected`
 Access Levels - Specification
 Class Data and Class Methods
 Adding Methods to Classes and Objects
 Special Global Variables
 Scope of Variables
 Built-in Classes
 The `Math` Class

The `NilClass` Class
`TrueClass` and `FalseClass`
 Built-in Class Hierarchy

Input and Output

Introduction
 Reading from the Standard Input
 Reading a Character at a Time
 Writing to the Standard Output
 Reading and Writing Disk Files
 Reading Files Using Iterators
 I/O With Command Line Commands
 Seeking About Files
`tell`
 Capturing Data About Files
 Processing Directories

Exceptions

Introduction
 Exceptions
 Handling Exceptions
 Multiple Rescue Clauses
 Exceptions are Classes
`ensure`
`retry`
`raise`
 Creating Your Own Exceptions
`catch` and `throw`

Modules

Introduction
 Using Core Ruby Classes
 Ruby Standard Library
`require`
 Search Path
 File Organization
`Load`
 Modules
`include`
 Mixins
 Using the Comparable Module
 Collection Classes
`yield`
 Using the Enumerable Module

Odds and Ends

Ruby Conventions
 Strings Are References
 The Selection Operator, `[]`
 Index Methods
 Stripping Whitespace Characters
 Bit Manipulation
 The `upto` Method
 Substituting
 Processing a Line at a Time
 Marshalling

Reflection
`grep`
 Classes are Objects
 Aliasing
 Testing
`Test::Unit`
 Testing Your Own Classes
 Freezing Objects

Course Description:

Groovy is a dynamic scripting and programming language for the Java platform. It combines the dynamic features of modern scripting languages such as Ruby and Python with familiar Java syntax. To quote one of the Groovy developers: "Groovy is what Java would have been if it had been created in the 21st century."

This course introduces the Java developer to the Groovy language. The course focuses on understanding the internals of how Groovy works in addition to understanding the Groovy language syntax. After taking this course developers will understand the Groovy syntax and be able to leverage existing Java classes within Groovy.

This class combines lecture with a unifying, hands-on experience, and open discussion that will help the developer quickly understand the benefits of Groovy and how to use the language.

Who Should Attend:

This class is for Java developers who wish to learn the Groovy scripting and programming language.

Prerequisites:

Participants should have taken the Software Development for Non-Programmers course or have a solid understanding of Java programming.

Benefits of Attendance:

Upon completion of this course, students will be able to:

- Write applications using Groovy.
- Understand how Groovy operates within the Java Virtual Machine.
- Incorporate existing Java classes and libraries within Groovy applications.
- Learn to add new methods and member variable to existing Java or Groovy classes dynamically.
- Understand the role of Closures within Groovy.
- Take advantage of Groovy's simplified object configuration syntax.
- Learn how to override operators for Groovy or Java classes.
- Understand the concept of Metaprogramming and how to leverage it to simplify application development.
- Explore Groovy's Regular Expression syntax for easily managing String processing.

Course Outline:**Language Overview**

What is Groovy?
 What Groovy can do
 Java programmers
 Script programmers
 Agile programming
 Installing
 Running Groovy scripts

Basic Syntax and Scalar Variables

Syntax rules
 Numbers - Integers, Floats, and BigDecimal
 Strings
 Double quoted
 Single quoted
 Here documents
 Slash quoted
 GStrings
 Operators

Collections

Lists
 Coding a closure
 The it parameter
 Passing multiple parameters
 Naming parameters (the -> operator)
 Maps
 Ranges

Flow Control

If statements
 The truth in Groovy
 Switch statements
 While loops
 For loops
 Exceptions

Classes

Defining classes
 File-to-class relationships
 Member variables
 Automatic getter/setter generation
 Default visibility

Safe dereferencing with ? operator
 Methods
 Optional parameters and default parameters
 Operator overloading
 Automatic constructor generation
 Initializing property values in the constructor
 The Closure Groovy class
 Coding a method that expects a closure
 Calling into the closure
 Passing parameters

Advanced Classes and Closures

Closures
 Using methods as closures
 Polymorphic closures
 Operator overloading
 Metaprogramming
 Discovering a class
 Discovering fields
 Discovering methods
 Method resolution
 MetaClass
 MetaProperty
 MetaMethod
 Pointers
 Method
 Field
 Calling methods that do not exist
 ExpandoMetaClass
 Categories

Regular Expressions

Regular Expression syntax
 The =~ operator
 The ==~ operator
 Common methods that use Regular Expressions

Builders and Slurpers

What are Builders and Slurpers?
 NodeBuilder
 MarkupBuilder
 AntBuilder
 Creating custom builders

Using the ConfigSlurper
 Writing a Slurper