CLIPCODE™

https://www.clipcode.net

Winter 2024 Curriculum

A series of one-day intensive on-site and remote training courses for senior knowledge professionals

"The mind is not a vessel to be filled, but a fire to be kindled" - Plutarch

CLIPCODE™

Clipcode Academy is pleased to announce its latest set of intensive one-day training courses aimed at senior knowledge professionals, in areas such as mathematics, software engineering and DevOps.

These courses are available for on-site presentation throughout Europe. Your team may request them "as is" or as part of a custom course, combined and extended to suit your specific requirements (e.g. you may design your own course by selecting modules of interest from different courses, or requesting new modules).

Our courses are based on the most up to date versions of the mathematical theories, programming languages, operating systems, runtimes, development tools, database engines and international standards that they cover.

To arrange on-site training, please phone us or e-mail academy@clipcode.com with a description of the type/location/duration of training you require and the number of attendees and their experience levels, together with your full contact details.

\<Design Your Own Course\>
\<You decide what goes here\>
This is our most popular course!

If you need on-site training for your team, and our existing courses match your specific needs – excellent! However, perhaps they partially but not completely match your requirements, or perhaps your team needs something more custom. In either case, then go ahead and design a training course to your own specifications. We would be willing to write and present it in your offices to your development team. To design your own course you identify the target audience & their existing skill sets, decide what new competencies they need and then select a coherent set of topics to be covered. You can treat each of our current courses as a menu of modules & choose as needed from each. Also you can identify new topics. Using existing specialist material we have, and where necessary creating new material specifically for you and combining this with appropriate selections from our standard courses, we can come up with the exact course you need. We work on the basis that if one client requires a course for a specialist area now, it is quite likely that others will require it in future.

We consider the new material we create as part of our ongoing curriculum development initiative and there is no additional charge for you beyond our normal fees.

	Contents of One-Day Training Course	
Target Audience \<You decide what goes here\> **Prerequisites** \<You decide what goes here\>	**\<You decide what goes here\>** \<You decide what goes here\> **\<You decide what goes here\>** \<You decide what goes here\> **\<You decide what goes here\>** \<You decide what goes here\>	**\<You decide what goes here\>** \<You decide what goes here\> **\<You decide what goes here\>** \<You decide what goes here\> **\<You decide what goes here\>** \<You decide what goes here\>

Mathematics Faculty

Mathematics / Foundations

- Fundamentals Of Mathematical Foundations
- Proof Theory
- Type Theory
- Category Theory
- Mathematical Logic
- Linear Logic, Adjoint Logic And Session Types
- Description Logic
- Computation Theory

Mathematics / Structure

- Abstract Algebra
- Linear And Multilinear Algebra
- Graph Theory

Fundamentals Of Mathematical Foundations
Terminology, Tour, Induction, Set Theory, Lambda Calculus, Complexity, Monad, Number Theory

Like ships passing nearby on a foggy night, each oblivious to the presence of the other, up to recently most mathematicians and software engineers ignore the work of the other. This course is about removing the fog, and letting developers see what can be achieved in production environments using really good ideas from modern mathematics. More and more ideas from mathematics are beginning to seep into the world of programming. Many software innovations – from deep learning to 3D graphics to modern programming type systems - are based on modern mathematics. All developers have studied some mathematics at college, so this course builds on that. It should be considered a refresher, with an emphasis on practical application, to bring everyone up to speed with the basics and be ready to explore more advanced topics.

We pay particular attention to how mathematical ideas are presented. For example, saying "a monad is just a monoid in the category of endofunctors" utterly confuses developers, whereas we prefer "monads are programmable semicolons" (just used to insert custom code between each statement) is much clearer, yet equally accurate.

	Contents of One-Day Training Course
Target Audience This course is aimed at modern developers who need a better grasp of how areas of mathematics can be practically applied to programming. **Prerequisites** It is expected attendees will have completed some mathematics training as part of their college education.	**Review Of Fundamentals** The language of mathematics (it is really not all Greek!) A mathematical object has certain properties and can be used in operations Mathematical structures are mathematical objects themselves that contains some arrangement of mathematical objects (often a set or similar, + something extra) **Terminology** Mathematical object (much broader use of 'object' term compared to programming) Symbols and (mathematical) variables Mathematical statement Proposition, expression, formula "a **proposition** is a statement susceptible to proof, whereas a **theorem** is such a statement that has been proven." (HoTT) **Branches of Mathematics ..** ..and their uses in software development Quantity/Arithmetic; Change/Calculus; Structure/Algebra; Space/Geometry Need to describe mathematical universes .. **Mathematical Whirlwind Tour** Quick tour of all of mathematics So many aspects to it – where do we start? We like "The Map Of Mathematics" Which new/unfamiliar parts we should use **Mathematical Induction** Definition by induction, in steps Base step (e.g. 0) is the starting point Induction step: builds on base (n) Example of deductive reasoning	**Set Theory** In the past, set theory was considered the most suitable approach to the foundations of all of mathematics More recent approaches (e.g. type theory, category theory) are better However, set theory is still an important area and worth studying Deductive system based on first order logic Law of the excluded middle (LEM) Axiom of choice, ... **Complexity Theory** Can a computational problem be solved? If so, how long will it take? Worst case scenario Big O notation **Monads** Isolating change in a non-changing world (think of carefully managed assembly lines - some steps make changes, others do not) **Lambda Calculus** The three terms (and how they work) – variables, abstraction and application Untyped vs. simply-typed Extensions (exceptions, recursion, ..) Currying (higher order functions) How Lambda Calculus is used in modern programming languages/type systems **Number Theory** Kinds of numbers (naturals, integers, reals, complex numbers) Broadening the scope (e.g. algebra) Specialist topics – e.g. Dedekind cut

Proof Theory
Natural Deduction, Props as Types, Sequent Calculus, Ordinal Analysis, Automated Prover, Verification

There are a number of fascinating sub-fields in proof theory and in this course we explore the best of them, starting with structural proof theory (e.g. natural deduction & sequent calculus), which has some very powerful features. We are particular interested in the junction of proof theory with programming. Many believe the future of programming will be significantly influenced by automated verification of correctness of code - but what does that mean and how can it be achieved? There are already open source projects pointing the way to much more extensive use of formal verification.

We are also curious as to what impact treating a proof as a mathematical object has, and using mathematics itself to explore a proof. Automated provers and proof assistants are improving in quality all the time and we see what the latest techniques offer. We also look at other branches of proof theory, such as proof complexity, ordinal analysis and provability logic.

Modern proof theory has huge potential to revolutionize our approaches to the rigor with which we judge statements made in mathematics and programming.

	Contents of One-Day Training Course	
Target Audience This course is aimed at mathematicians and modern developers who need a better grasp of proof theory **Prerequisites** Good foundational mathematical education along with some programming experience. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.	**Overview of Proof Theory** What is proof theory? Many branches – structural proof theory, provability logic, proof mining, automated theorem proving, ... Structural proof theory includes natural deduction/sequent calculus/hilbert A proof as a mathematical object- can be manipulated and reasoned about like any other mathematical object **Formal Verification** Mathematically proving that code works in all circumstances will always be more desirable than unit testing it for known scenarios **Natural Deduction Intro** Judgment, evidence and witnesses Depending on what kind of logic that interests us, differing judgments needed Introduction rule Elimination rule Multiple premises and a single conclusion **Advanced Natural Deduction** Alternative representation styles Assumptions / context / use of turnstile: ⊢ Real world ND usage: defining a language such as WebAssembly **Proposition As Types** Relationship to Lambda Calculus Propositions as types Proofs as programs Normalization as evaluation of programs Nirvana: code is (provable) logic is code	**Sequent Calculus** A sequent consists of propositions to the left (ANDed), a turnstile and propositions to the right (OR) Cut elimination Importance for linear logic Supports multiple premises AND multiple conclusions (unlike natural deduction) **Automated Proving** Proof assistants Automated theorem provers **Proof Verification** Deciding if a proof is correct Comparing proofs **Proof Complexity** Determining complexity critical for practical automation Identifying the number of steps needed for a valid proof As problems get larger, proof size explodes **Ordinal Analysis** Created by Gerhard Gentzen, ordinal analysis helps with consistency of proofs An infinitary proof calculus Upper bounds & lower bounds **Provability Logic** 'it is proved that ..' Relationship to modal logic The Gödel-Löb logic of provability **Project: Using Proof Theory** We conclude this course by exploring how proof theory can help us create a modern programming language

Cubical Computational Type Theory
Theory of Truth, Types, Equality, Identity, Universes, Univalence, HITs, Kan Filling, Cubical Type Theory

The related trinity of type theory, mathematical logic/proof theory and category theory forms a modern foundation for all of mathematics and computation.

 A type is a precise mathematical specification of behavior. An element of a type satisfies its specification. Both a type and its elements are programs, subject to evaluation. Elements evaluate down to values and types evaluate down to canonical types [Martin-Löf] or type values [Harper] (same idea). Application developers experienced with mainstream programming languages will already understand elements (objects) are subject to evaluation, but will find types being so to be new and significantly more expressive. Type theory supports the idea of indexed families of types-the index being a value (also known as dependent types). This will also be a new idea to most developers. By introducing these ideas and lots more, modern type theory goes far beyond what we see as type systems for regular programming languages. This course focuses on the most promising of the latest type theories, known as cubical type theory- a computation-friendly approach to new ideas such as univalence & higher inductive types. The Redtt project is an interesting implementation.

	Contents of One-Day Training Course
Target Audience This course is aimed at modern developers who are specifically interested in modern type theory and how it can be practically used in computation **Prerequisites** Good foundational mathematical education along with some programming experience, as we include exploring type theory from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.	**A New Foundation** **Additional Concepts** Type theory The meaning explanation Category theory Functionality (in a type theory sense) Mathematical logic/proof theory The hypothetical judgment How they relate to each other (Shulman) Recursion Constructivism **Equality and Identity** **Type Theory Overview** Introduction to equality in type theory Think of modern type theory as a highly Variations of equality type (exact, ..) expressive programming language useful Identifications for mathematics (+everything based on it) **Univalence** A theory of truth – based on Martin-Löf's Initially considered in terms of Homotopy ground-breaking paper: "Constructive Type Theory (HoTT) mathematics and computer programming" That uses an axiom – not desirable from a **Basic Concepts** computational viewpoint – why? Type So other approaches explored ... Element **Cubical Type Theory - Intro** Evaluation This is the cutting edge type theory Judgment Simple intro Witness Uses points, lines, planes, cube, n-cube Intuitionistic Type Theory The idea of paths **Simple Types** Transporting Defining a type **Cubical Type Theory - Details** Defining an element Kan filling, hcom, coe Look at bool and nat(ural number) Higher dimensionality **Common Types** Cartesian variant Type **Higher Inductive Types (HITs)** Function HITs as generalization of inductive types Sum Uses in various forms of construction Product Diagonals Top / Bottom **Redtt Proof Assistant** **Dependent Types** Redtt is an open source implementation of The idea behind dependent types cubical type theory – let's explore how it Creating an indexed families of types works

Category Theory
Category, Category Object, Maps And Composition, Functor, Natural Transformation, Adjoint

Category theory is a foundational area of mathematics that examines in a uniform manner mathematical structures and their mappings. In category theory, a category is a mathematical universe. A category is populated by category objects and there are mappings (also called morphisms) between these objects.

A category is itself a mathematical object. There can be mappings between categories, known as functors, and even mappings between such functors, known as natural transformations. Categories are abstract representations of concepts from other areas of mathematics.

It is best to think of the goal of category theory is to describe an abstract multiverse, containing one or more universes, with mappings within each universe, and between them. From this surprisingly simple core of constructs, a rich description of much of modern mathematics can accurately be built.

This course helps mathematicians and software developers gain an appreciation of what is category theory, both basic concepts and more advanced capabilities, and to see see how it can be practically applied in real-world situations.

	Contents of One-Day Training Course	
Target Audience This course is aimed at mathematicians and software engineers interested in learning about category theory **Prerequisites** Good foundational mathematical education along with some programming experience, as we include exploring category theory from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.	**Category Theory Overview** Why the interest? What is a category? Practical applications Examining structure-preserving mappings between objects **Basic Ideas** Introduction to the basic constructs A category object is a mathematical object (to begin with, think of it as a set of elements) - so is not the same as an object in typical OO programming Think of a mapping (morphism) between category objects as a relation Maps can be composed – so if we have map f and map h, we compose them as h o f (read as "h follows g") Such compositions form paths **Mathematical Definition** A category is defined as a collection of category objects, with maps (with domain and codomain) between them Each object has an identity map Two important laws: * Identity Law governs identity map use * Associative Law: (h o g) o f = h o (g o f) **Functors** Functors are mappings between categories Structure-preserving Endofunctors are mappings from a category object to itself -"endo" (as in "endoscope") means look inside oneself Left/right adjoint functors (as in adjoining)	**Exploring Sample Categories** Set, Ring, Monoidal, Group, 2-category, custom categories **Advanced Category Theory** Natural transformations A deeper look at a variety of morphisms Limits and colimits Universal property Initial and terminal object Presheaf **∞-groupoid** "Sets in the next dimension are groupoids" Groupoid builds on group in group theory A groupoid is a category where each morphism is an isomorphism ∞-groupoid generalizes groupoids k-morphisms and equivalences **Cartesian Closed Categories** Corresponds to lambda calculus Mapping to and from the lambda calculus "A CCC is a category that has an exponent and a product, and is closed over both. The product is an abstract version of cartesian set product; the exponent is an abstraction of the idea of a function, with an "eval" arrow that does function evaluation." [link] **Triumvirate** Role of category theory as one member of the triumvirate that includes type theory and mathematical logic "Roughly speaking, a category may be thought of as a type theory shorn of its syntax" [link]

Mathematical Logic
Logic as Foundation, Propositional, First Order, Higher Order, Models, Temporal, Modal, Kripke

Mathematical logic permeates all of mathematics and programming.

Building on simple propositional logic, a host of richer logics can be constructed to target different needs. For example, First Order Logic is the deductive system used by set theory and is also the basis for Description Logic. Higher Order Logics have richer predicates though come with added complexity.

We need to look at what we can represent in logic and what valid claims we can make about such.

Logic can be qualified by time, by modality or other techniques. More advanced logics are based on this common foundation. It is useful to consider these logics as a set of building blocks which can be rearranged to suit specific needs. We are interested in building our own logics. We also want to use mathematical logic in real projects and explore how best to do this.

By the end of this course it will be clear that a good understanding of mathematical logic underpins a good understanding of all of mathematics and programming.

	Contents of One-Day Training Course	
	Logic Fundamentals	**Structural Rules**
	Defining and using a formal logic	* Weakening
	Review of logic as branch of mathematics	* Contraction
	Part of foundations	* Exchange
	List of logics	* Associativity
	Good understanding of logic helps with understanding everything above it	Being selective - substructural
Target Audience		**Modal Logic**
This course is aimed at mathematicians and modern developers who need a better grasp of how mathematical logic can be used in practice	**Propositional Logic**	Modality
	Conjunction	Modal terms and their impact
	Disjunction	In some way we wish to qualify or restrict
	Negation	a logic statement
	Conditional	Modal operators
	Truth tables	**Kripke Semantics**
	Predicate (First Order) Logic	Usefulness
	Builds on propositional logic	The Kripke model
	∀ means "for all"	Soundness of a modal logic
	∃ means "there exists"	Kripe Frame
Prerequisites	Quantifiers	**Temporal Logic**
Good foundational mathematical education along with some programming experience, as we include exploring logic from a computational viewpoint. Attendees can select which programming language they wish to use, as all concepts will be developed from first principles.	More advanced symbols	Logic that is in some way time-based
	How first order logic is used in set theory	Now, future, past, until, while, always, …
	Higher Order Logic	Temporal operators
	Extra quantifiers	More fine-grained representation of things: how they are in the real world (beyond being simply always true)
	Predicates themselves having parameters	
	What variables range over (sets of sets)	
	Quantifying over sets	**Custom Logic**
	Additional semantics	Why would we want to define our own logic?
	Model Theory	Steps involved
	Mathematical models	How to create and use
	An interpretation gives meaning to symbols in a formal language	**Project**
	When is an interpretation a model?	Practical use of a variety of mathematical logic in a production setting
	Interpretation function	
	Domain	Considerations and observations

Linear Logic, Adjoint Logic And Session Types
Cut Elimination, Substructural Rules, Linear Logic, Adjoint Logic, Message Passing, Sessions, Scribble

[Sample: link hub] Data types have played a pivotal role in software engineering for the last 50 years and many believe session types will have an equally profound effect for the next 50. Software platforms are getting more complicated, larger, have more moving parts and errors are more costly to remedy. It is clear that the software engineering community needs more robust tooling and modern mathematics can certainly help.

A number of potentially very useful mathematical logics have already been defined, the most interesting from a communications and concurrency perspective is linear logic. Multiple logics can be combined using modal operators based on adjoint logic. These can be used where we need to mathematically represent and accurately reason about processes/threads and their concurrency and message passing constructs and use-once or other richer semantics.

Due to its many practical usage scenarios, the juxtaposition of mathematics, concurrency and communication is a very active research area. We are beginning to see these ideas have an impact on new specialist tooling and extensions to existing frameworks.

Contents of One-Day Training Course	
Target Audience Mathematicians and software engineers interested in taking a rigorous mathematical approach to concurrency and communication **Prerequisites** Good knowledge of the fundamentals of mathematical logic.	**High Level Goals** Deadlock freedom during communication Distributed garbage collector Shared memory at scale Mathematically sound concurrency **Preliminaries** Cut elimination Sequent calculus What is a sequent? The idea of a process calculus (e.g. π-calculus) **Evolution of Propositions As Types** "propositions as session types, proofs as processes, and cut elimination as communication" [link] **Substructural Logic** Substructural logic – tweaking with the structural rules, what effect that brings? **Linear Logic** What does it describe? Mathematically describe a process (a thread of activity) as a series of steps Mathematical duals In contrast to normal mathematical logic, which focuses on truth (constant), linear logic focuses on resources (e.g. use once) Highly useful for writing deadlock-free code and communication specifications **Adjoint Logic: Intro** Fundamentals of adjoint logic Practical uses Review of modal operators Adjoint pairs of modal operators	**Adjoint Logic: Details** Relationship to weakening and contraction Combining a variety of logics, such as LNL, linear, s4, lax, ... Modes: * linear, * affine, * strict, * unrestricted **Message Passing** Interesting: message passing interpretation Using adjoint logic to structure messaging **Session types** Practical uses of session types in modern programming - intro type discipline to code Managing multiple communicating and concurrent sessions Binary and multi-party **Scribble** "Scribble is a language to describe application-level protocols among communicating systems. " [link] How Scribble works Generating Finite State Machines (FSMs) on both sides **Usage** Some of these ideas are being to be used in real products and development tooling Survey of what is available and how to use **Project** Expanding on practice topics explored earlier with a project looking at practical deployment of these ideas in production

Description Logic
Representation, Reasoning, Tableau Algorithm, Subsumption, DL Extensions, Querying, Ontologies

Description Logic is the mathematics of the knowledge graph.

Description Logic is the discipline of mathematics concerned with knowledge representation and reasoning. There are many ways to represent knowledge as data but where Description Logic excels is doing this in such a way that greatly facilitates reasoning.

DL builds on a number of areas of mathematics such as the decidable fragment of First Order Logic and model theory, and pays attention to complexity theory.

There are a variety of Description Logics (DLs), and we compare and contrast a number of these. More advanced DLs feature richer expressivity, but also come with increased complexity in reasoning (especially for large knowledge bases), so it is important to understand the tradeoff.

A good understanding of Description Logic is essential when working with graph stores, reasoners and semantic query languages. DL provides the mathematical formalism that underpins the semantic web, the W3C OWL Language and the reasoners that process it.

	Contents of One-Day Training Course	
Target Audience This course will be of keen interest to mathematicians and software developers who wish to understand Description Logic - the mathematical foundations behind the Semantic Web and W3C specifications such as OWL, RDF and SPARQL. **Prerequisites** Good foundational mathematical education along with some programming experience, as we include exploring Description Logic from a computational viewpoint. Attendees can select which programming language they wish to use in the labs, as all concepts will be developed from first principles.	**Description Logic Overview** Relationship to first order logic Description based on concept, role and individual plus operators TBox and ABox Importance of inferencing Constructing a mathematical model **DL Basics** Defining a simple DL How to represent knowledge using it What services could we layer on top of it? **Model Theory** An interpretation is a mapping for concepts, roles etc. and is a model if certain conditions hold Relationship between DL & Model theory **Knowledge Services** Subsumption / Consistency Inferencing Satisfiability Querying **Tableau Algorithm** More expressive DL needs richer reasoner Outline of tableau algorithm Variants DeMorgan's theorem **DL Extensions (extra letters)** Nominals Cardinality restrictions Inverses Temporal Extra role constructors Concrete domains	**Family of Logics** SHOIN (OWL DL) SROIQ (full OWL2) SHIF (OWL-Lite) … lots more **Resources** Tooling Frameworks Servers Reasoners **Advanced Topics** Automata as a specialist alternative to the tableau algorithm Study of complexity on relation to DLs Scalability (very large knowledge bases) Fixpoints **Use of DL in OWL** The W3C semantic web defines the Web Ontology language (OWL) which is based on description logic The DL terms (individual, concept, role) map to OWL terms (individual, class, prop) **Ontologies vs. SQL databases** There are similarities & differences (TBox = schema, ABox = data) Open vs. closed world assumption Incomplete information Role that reasoning plays Unique name assumption **Query Answering** How it works (query processing) Covers given and inferred knowledge Re-writability

Theory Of Computation
Automata: FSM, Inputs, Outputs, DFA, NFA, Computability: Turing Machines, Complexity: BigO

An automaton (plural: automata) is a logical model of a machine that, based on input events, transitions from state to state. To describe an automaton we need to identify its states, the set of acceptable inputs and the expected outputs, and describe how transitions work. There are a number of optional additional features to constructing automata and these can add a range of extra capabilities.

Automata theory is used throughout mathematics and programming (e.g. compilers, protocols). It's sometimes so natural that users are often unaware of its presence.

Computability theory shows how an abstract machine can be subjected to mathematical reasoning and certain important characteristics can be reliably proven. We explore the best known of these – a Turing Machine. Complexity theory helps classify the degree of difficulty (from trivial to impossible) there is in solving a given computational problem.

A clear understanding of the theory of computation helps everyone on a team have a richer appreciation of how automata, computability theory and complexity theory can be beneficial to a product's architecture.

Contents of One-Day Training Course	
Target Audience This course is aimed mathematicians and software developers who wish to become familiar with important aspects of how mathematics plays a foundational role in computation. **Prerequisites** Attendees need a good understanding of mathematics and software programming.	**Overview Of Automata Theory** Practical uses of automata Overview of automata theory Deterministic vs. non-deterministic How different automata vary **Types of Automata** In increasing order of complexity: * Finite state machine * Pushdown automata * Linear bounded automata * Turing machine What more complex automata brings **What is Needed to Build** States Inputs – what drives transitions Outputs – result of transitions Transitions **States And Transitions** Identifying states Optionally - identifying initial / final states May be more than one Transition function **Deterministic Automaton** A given sequence of inputs will result in a given set of state transitions A set of states A set of inputs the next state function the final predicate **Non-Deterministic Automaton** Impact of non-determinism Transition relation Converting a NFA to a DFA	**Specialist Automata Topics** Acceptance conditions Automata with an infinite number of states Cooperation between multiple automata Relationship to computational theory Asynchronicity **Computability Theory** What is computability & recursion theory? Model of computation - mathematically describing computation Examining the properties of computation Reverse mathematics **Deep Dive: Turing Machine** What is a Turing Machine? How does it work? What does its operation demonstrate? Understanding this abstract machine brings many benefits **Intro to Complexity Theory** Computation with large numbers of steps and states can have performance issues Specifically for these, need to consider variation of approaches and how to measure complexity **Advanced Complexity Theory** Trying to estimate amount of resources needed for particular compute workload Exploring the limits of computation Can a problem be solved at all? **Project** Use of theory of computation in a non-trivial project to show its benefits in a practical setting

Abstract Algebra
Groups, Rings, Fields, Lattices, Modules, Algebras
Role of Algebraic Structures, Applications

With algebraic structures we try to model characteristics of certain mathematical arrangements of elements and operations upon them. We often find the same structures appearing in different areas of mathematics and by more deeply understanding properties of such structures in the abstract we can share insight we discover in one area across different application areas. We see the varied algebraic structures can be used to represent significant amounts of modern mathematics. For example, what has been discovered about common number arrangements (natural numbers, integers, real) actually can be applied more widely.

The main families of algebraic structures that we explore in this specialist course are groups, rings, fields, lattices, modules and algebras. We say "families" since these structures have multiple variations and depending on what we are trying to describe and reason about, we choose differing representations.

Some more specialist representations explore combining different structures (such as Galois Theory, which combines fields and groups). Sample applications include the study of quaternions.

	Contents of One-Day Training Course	
	Fundamentals of Algebraic Structure	**Field**
	Overview of algebraic structures	A field is a set with operations for:
	Group, ring, field and more	addition, subtraction,
	Order and partial ordering	multiplication and division
	Sequence and maps	A commutative ring
	Usage scenarios for algebraic structures	**Finite Fields (Galois Fields)**
Target Audience		A finite number of elements
Those interested in algebraic structures and their practical applications in a number of areas.		Impact on fields
	Group	Uses in areas such as cryptography
	A (mathematical) group is a set with a binary operation and certain properties (related to identity, inverses and associativity)	**p-adic Fields**
		"*For p any prime number, the p-adic numbers .. are a field that completes the field of rational numbers. As such they are analogous to real numbers.* " [link]
	Abelian group (adding commutativity)	
	Semi-groups	**Lattice**
	Monoid	Adding the $\wedge$ and $\vee$ operations and $\top$ / $\bot$
	A group with an identity element but (perhaps) without inverses	As a kind of poset
		Bounded lattices
	e.g. natural numbers (with 0) as a monoid under multiplication, under addition	**Module**
Prerequisites		Module over a ring
Attendees need a good foundation in mathematics with some previous algebra experience.	**Groupoid**	As an abelian group
	A category where every map is invertible	Representation of a group
	Similar to group but with binary operation replaced by partial function	Module over a monad
	Groupoid compared to equivalent relation	**Algebra**
	Partial equivalent relation (PER)	Distinct terminology from algebra theory
	Ring	Algebra over a field
	A (mathematical) ring is a set with two binary operations (similar to addition and multiplication)	Associative and non-associative algebras
		Quaternions as an algebra
	Representing a ring as a triple	**Practical Applications**
	Properties of rings	Where might algebraic structures be applied for practical benefits?
	Ring theory	Exploring sample applications

Linear And Multilinear Algebra
Concepts, Vector Spaces, Matrices, Determinants, Eigenvectors, Tensors, Applications

"*Linear algebra is the study of linear sets of equations and their transformation properties*" [link]. We see linear algebra having widespread practical uses in fields such as AI, computer vision, 3D graphics and animation, along with many areas of engineering, science and the financial markets.

In this course we explore many aspects of modern linear algebra, starting with fundamental ideas building up to how they can be practically used. We begin with vector spaces which provide an excellent representational structure for what we need to investigate.

We cover matrices and their many characteristics in depth, as they have been widely studied over the years and many useful theorems have come to light. This course also investigates multilinear algebra including the use of tensors, which we see having growing importance. Hence the use of the term "tensor" in product names (e.g. Google's TensorFlow, and TPUs; and NVIDIA's Tensor Cores). What are tensors and what are they considered so useful in various fields? This course concludes by exploring in depth how linear algebra can be implemented in programming frameworks.

Contents of One-Day Training Course	
Target Audience This course is aimed mathematicians and software engineers who wish to become familiar with all aspects of linear algebra. **Prerequisites** General understanding of the fundamentals of mathematics and some programming experience.	**Overview of Linear Algebra** Theoretical and practical applications of linear algebra Role of linear algebra in AI, 3D, elsewhere Number of ways of representing linear algebra – why we prefer vector spaces **Vector Spaces** A vector space over a field whose elements are scalar, with a set containing elements which are vectors Vector addition Vector multiplication Vector spaces as a way of working with sequences, functions, polynominals and matrices Linear transformation **Axioms for Vector Spaces** Axioms for vector spaces: * associativity * identity * inverse * distributivity, .. **Matrices** Representing and working with matrices Matrix multiplication Matrix inverse Square matrix Basis **Determinants** Determinant of a square matrix Invertibility Cramer's rule Determinant computations	**Eigenvectors & Eigenvalues** Reasoning about transformations Uses for stability analysis etc. Matrix diagonalization Left and right eigenvectors "eigen" means "proper" in German Calculations with eigenvectors An eigenvalue as a scale factor of an eigenvector **Multilinear Algebra** Linear algebra constructions for tensor products of vector spaces p-vectors Multivectors Practical uses of multilinear algebra **Tensors** A linear mapping from one set (e.g. of scalars, vectors, tensors) of algebraic constructs to a different one Role in TensorFlow Tensor calculations lend themselves to hardware implementations for massively improved performance **Linear Algebra Programming** Dense and sparse matrices – and their differing characteristics under different kinds of loads Gaussian elimination Applying linear algrebra to datasets **Project** Exploring the implementation of linear algebra concepts in a programming framework and its use in a practical project

Graph Theory
Representation, Storage Alternatives, Traversing, Searching, Operations, Graph Drawing, Project

A graph is one of the most versatile structures in mathematics with widespread research & practical uses. We see use of graph concepts in areas ranging from the knowledge graph, the social graph and in organization-specific graphs (e.g. Microsoft Graph). Devs are used to graph terms such as the "object graph" or the "call graph". Graph databases are becoming very popular. Anywhere you see the work "network" in industry there is a mathematical graph lurking underneath. Because of its popularity, we even see use of the "graph" term where it shouldn't be – e.g. GraphQL (which is neither graph-based, nor indeed, a query language).

This course covers all aspects of graph theory, from simple representation, to traversal and search, to transformations, to display. Starting with simple edges and vertices (with storage as either a matrix or a linked list [preferred for sparsely populated graphs]), we investigate how to build, transform and present graphs.

We explore how to efficiently handle large graphs with a keen interest in high performance. This course also covers graph drawing (a surprising complex topic in its own right). We conclude with a project to build a graph engine.

Contents of One-Day Training Course	
Target Audience This course is aimed mathematicians and developers who wish to become familiar with the theoretical and practical usage of graphs in a variety of scenarios. **Prerequisites** Attendees need a good foundation in mathematics and programming, as this course will be covering graph-related ideas from both disciplines. Attendees need to be familiar with one programming language. Any will do, as in the hands-on labs they will be developing a graph library in that language from first principles.	**Overview of Graph Theory** Part of discrete mathematics No "top" or root Overview of graph concepts Vertices and edges **Introduction to Graphs** $G = (V, E)$ where V is a node set and E is an edge set Simple graph Multi-graph Common operations on graphs Basic terminology Directed (digraphs) vs. undirected graphs **Traversing A Graph** The visitor pattern Ensuring each node is visited Weighted graph (e.g. assign a "cost" to each edge, to influence selected path) **Searching A Graph** Breath-first search Depth-first search Dealing with cycles **Graph Operations** Graph composition: merge points to be based on identity of edges Subsumption Node selection Edge contraction **Types of Graphs** Connected graph Bipartite graph Complete graph Regular graph	**Graph Properties** Shortest path Minimum spanning trees Traveling salesman **A Tree As A Kind Of Graph** Every tree is a graph, but not reverse Binary search trees Subtrees **Graph As Network Flow** Networks appear throughout engineering, science, business and daily life – how to best represent as graphs Max flow / min cut theorem Interacting with networks as graphs **Graph Data Structures** Adjacency matrix vs. adjacency list How to optimize for large graphs Indexing Improving storage on hard disk **Graph Databases** Review of how graphs are handled by a graph database Querying possibilities Sample usage **Graph Drawing** How to automate placing graph constructs on a planar surface The crossing number How to improve automated layout **Project** Creating a custom graph engine to store and interact with large-scale graphs efficiently

Technology / Code School

Modelset

- Domain Driven Design
- Semantic Models
- UML
- Design Patterns
- Advanced Data Structures

Assembly

- x86-64 Assembly
- ARM Assembly
- WebAssembly And WASI

IR

- .NET IL

Compiler

- C 23
- C++ 23
- C# 12
- TypeScript
- The Java 22 Language
- Python
- The Go Language
- OCaml Functional Programming

Proof Assistant

- Coq Proof Assistant

Shell

- PowerShell 7
- Bash

Domain Driven Design
Ubiquitous Language, Entities, Aggregates, Events, Services, Repositories, Contexts, Strategic Design

Every enterprise application has a domain – the actual area where the application delivers business value to real users. The database, user interface framework, messaging infrastructure, etc. are just tangential to what the application is really about. In the past, engineering teams concentrated so much on SQL database tables and GUI form layout that the handling of the domain was swamped and lost in the mix.

In contrast, modern software engineering teams rightly place much greater focus on the domain layer of the project and incorporate domain driven design as a central pillar of their project development strategy. Its substantial benefits become very clear on larger projects and on projects that evolve over many iterations. This course explores all the patterns that underlie domain driven design with the goal that at the end of it, attendees will be fluent in DDD and can move from being participants in, to contributors to future projects that incorporate an important domain model. What's above (e.g. the UI) the domain model and what is below (e.g. database, messaging) may well change frequently over iterations, but the domain layer itself will have longevity, so it is extremely important to get it right.

Contents of One-Day Training Course

Target Audience

Object-oriented developers, architects and product managers who wish to build robust domain models as the central core of their applications.

Prerequisites

Good all-round experience of software engineering and product development

Overview
What is domain driven design?
What is its role in the larger software
   development ecosystem

Layering
Interaction (or UI) layer
Application (or command) layer
Domain layer
Infrastructure layer
Think of the domain as the middle of a
   sandwich rather than a slice of a pyramid

Ubiquitous Language
"All singing from same hymn sheet"
Identifying a common terminology and set
   of meanings that all stakeholders can use
Language of the domain (so non-techies
   can easily understand it)

Entities and Value Objects
Important role of identity
What do we need to identify (and how)
How do we attach values to identities

Aggregates
Boundaries and associations between
   groupings of entities and value objects
Controlled access

Domain Events
State changes
What is happening inside the domain
   model and exposing this to outside

Factories & Services
Constructing and supplying entities
Segregating specific responsibilities
Integration with dependency injection

Repositories
Connecting to a database (e.g. ORM)
   with an aggregate access service
Creating and calling queries
Role of testing

Bounded Context
What is is inside and outside the scope of
   a domain model
Importance of boundaries & multiple
models (good fences make good neighbors)

Supple Design
Intention revealing interfaces
Side-effect free functions
Assertions
Conceptual contours
On-going model evolution

Strategic Design
Core domain
Segregated core
Abstract core
Dependencies – managing relationships
   between large project subsystems

Large-Scale Projects
Review of layering
Published language and internals
Extensibility
Flexible architecture for longer lifecycles
Handling large systems

Project
The combined use of many domain driven
   design ideas inside a larger project –
   including creating the domain model
   and its use from other layers

Semantic Models
Domain / UI / Entity Data / Security / REST / Learning / Extensibility / Admin / Deployment Model

[Detailed Intro] Imagine a dev team has mastered core technologies and building on that solid foundation, now needs to design a next generation solution – how do they go about it? If you team is in this situation, this advanced course is what you need. Looking beyond the technologies, this course examines how to develop cutting-edge solutions that are of production quality, commercially competitive and feature all the "-abilities" your customers demand (scaleability, testability, manageability, usability, reliability, ...). The central idea is that software development revolves around a series of semantic models (domain, entity data, user interaction, security, admin, deployment, etc.) and these are grounded in a "single truth" of the source code which ensure all models work together.

For a new solution, we need to examine what is required to design and build each of these models so that the integrated end result delivers upon the expectations. Modern developer technologies allow us to be highly creative and more productive in how we go about designing web solutions. However, using them all together is somewhat of a challenge and this course carefully explores how to proceed.

	Contents of One-Day Training Course
Target Audience Advanced software architects and senior software engineers who wish to design cutting-edge solutions using the latest design ideas. **Prerequisites** It is essential that attendees have a good all-round experience of the technologies they wish to use.	**Model Driven Development** Q: Assuming we know the very latest technologies and APIs, how do we go about designing next generation solutions? A: Using a series of semantic models **Domain Model** Domain Driven Design Handling complexity Central role of the Domain Model Evans' patterns for Domain Driven Design – Ubiquitous Language, Entities, ValueObjs, Intention-Revealing Interfaces, Services, Repositories, Layered Arch, ... **Entity Data Model** How we bind objects and relational data Creating data models & abstractions Using entities in other parts of app **User Interaction Model** Thinking past the widgets and visual layouts, we concentrate on what the user is really trying to achieve Task-focused design Making available functionality as needed Fluid interaction; paradigm selection **Security Model** Deciding what needs to be protected and how; verifying security of solution How to explain your solution's security model to others (e.g. security auditors) **REST Model** Representational state transfer can be used to selectively expose the Domain Model and Entity Data Model to remote clients **Deployment Model** Get the bits deployed is important (on average, 50% of support calls relate to this) Treating deployment as first class feature **Administration Model** Different levels of users Administrative features **Support Model** What happens when a user has a problem How can software help with support Instrumentation for support **Education Model** Often an afterthought, the education model is how users/admins learn to use the app It is crucial to on-ramping of new users and the amount of (costly) support they need **Test Model** Unit tests, load tests, security tests, ... Dependency injections and mocking Managing the testing process **Intelligence Model** Extracting actionable results from various information stores and making it promptly/easily available **Extensibility Model** Exposing an API to enable third parties extend your solution – approaches **Reference Architecture** Exploring a reference architecture built on ideas covered in this course. **Project** Attendees will work in teams to develop slice of a suitable project (they choose)

Unified Modeling Language
UML Views of a Project, Diagramming Notation, Use Cases, Classes, Relationships, Behaviors, States, OCL

UML is used by object-oriented designers to unambiguously specify an OO design, to discuss it with other designers and to communicate it to all stakeholders - developers, end-users and management.

We can think of UML as a graphical design language for object oriented software. It has become the graphical lingua franca of object design, supported by most design/development tools and understood by most software professionals. To discuss modern software among a team of stakeholders, we need to describe objects – their internal static information, how they relate to each other, how they behave dynamically (both intra- and inter-object) and how they are delivered to end-users. Software systems are becoming increasingly complex and at the same time there is persistent pressure to decrease development costs and time scales, and improve quality.

UML is the key to succinctly describing correct software architecture, and this in turn is the cornerstone of successful projects. A very important point is such design needs to be sufficient without being excessive.

	Contents of One-Day Training Course
Target Audience Experienced object-oriented architects and developers who need a detailed understanding of all aspects of UML. **Prerequisites** A good understanding of object-oriented principles and previous OO design & programming experience.	**UML Introduction** "The UML is a language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system" (Booch et al) **UML Views** There are five ways of looking at systems: User, Design, Process, Implementation and Deployment views UML diagramming notation Class name, operation and attribute, interface, component, package, note, state, state transition, event, action, etc. **UML Diagram Types** Use cases, class, object (instance), sequence, collaboration, statechart, activity, component, deployment **Classes and Relationships** The structural (static) aspects of a project Completely defining operations/attributes Generalization **Advanced Classes and Relationships** The four ways classes relate to others Aggregation / composition **Interfaces/Components/ Deployment** Functionality as components Collaborations **Dynamic Behavior** The dynamic aspects of a project Interaction among a set of object instances Finite state machine within an object (statechart diagrams) General system activity (activity diagrams) **UML For Database Design** The Persistent tagged value Storing objects in a relational database Generating database schemas from UML **Object Constraint Language** OCL enables tighter specification by adding detailed constraints to elements within object models **Round-Trip Engineering** Converting between UML and languages such as C++, Java, C# **UML and …** Multithreading Networking User Interface Frameworks **UML and Agile Development** Requirements, Analysis, Design, Implementation, Test, Delivery – all progressing in parallel **Design: sufficient, not excessive** How much/little design does an agile actually project need? How to go about creating it? UML as a form of sketching for devs **Complete Example** Walk through of creating a set of UML diagrams for the lifecycle design of a modern multithreaded HTTP web server

Design Patterns

Pattern Template, GoF Patterns, Applying Patterns, Advanced Patterns, Anti-Patterns, Pattern Extraction

[Detailed examples] Design patterns capture successful design experience for later reuse by the original/other designers. They capture solutions that have evolved over time, in a concise and easily applied fashion. Typically they are not the "first attempt" at solving a problem – but rather the result of an iterative design process by experienced designers (who have benefited from hard-learned lessons of previous projects). Teams are under tremendous pressure to produce higher quality software at lower cost. One option is to ship the work to cheap offshore development partners. A better option is to use smaller, but much higher skilled teams - who will compete by working more effectively to build the software. Such advanced teams will be trained in design patterns and hence can aggressively leverage them to retain a productivity and quality advantage over lesser skilled competition. As software projects increase in scale, cost and complexity, and involve more inter- and intra-company relationships, there is a need to adopt techniques such as design patterns, to ensure use of best practices in design issues. The goals of this course are to look at a range of design patterns, to examine how to apply them in your own projects and explore how to create your own pattern catalog.

	Contents of One-Day Training Course	
Target Audience This course targets senior software engineers and architects who need to leverage design patterns correctly in their own projects. **Prerequisites** Attendees require good OO knowledge and plenty of development experience.	**Overview of Design Patterns** Simple and elegant solutions Applying the concept of generics from programming to software architecture Catalog of design patterns **Defining a Design Pattern** Documentation template Important GoF fields (intent, motivation, applicability, structure, participants, collaborations, consequences) Additional fields by others **GoF Creational Patterns** Abstract Factory, Builder, Factory Method, Prototype, Singleton **GoF Structural Patterns** Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy **GoF Behavioral Patterns** Chain Of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor **Applying Patterns** Applying patterns in your projects Implementing patterns in code Evolution of project over time **Design Patterns and …** eXtreme Programming Aspect Oriented Programming (AOP) Agile Development Documentation **Server Platform Patterns** Patterns to satisfy competing demands	Pooling, tuning, managing, sharing, distributing, extending, scheduling **Concurrency/Network Patterns** Wrapper, Component configurator, Interceptor, Extension interface, Reactor, Proactor, Async completion token, Acceptor-connector, *-locking, Active object, Monitor object, Half-sync/halfasync, leader-follower **Enterprise Integration Patterns** Gregorgrams Message exchange, channels, headers **Security Patterns** How to correctly design security features into your software systems Secure channel, Session, Role, Checkpoint, Single access point, Full/limited views **Anti-Patterns** Anti-patterns "let you zero in on the development detonators, architectural tripwires and personality booby-traps that can spell doom for your project" (Brown) **Pattern Extraction** Finding patterns in your own projects Effort involved in extraction Optimizing and generalizing patterns **Custom Pattern Catalog** How development teams can build up a catalog of patterns for their own needs **Project Using Patterns** An advanced project whose architecture uses an assortment of design patterns

Advanced Data Structures
B-Tree, Splay, Priority Queue, Complexity, Searching, Sorting, Hashing, Algorithms, Implementation

In this specialist course we examine modern data structures from a mathematical viewpoint.

We explore how to correctly design modern data structures, how to implement the algorithms that interact with them, memory management, traversal, item searching and sorting, merging and splitting, and lots more. We use both imperative and functional programming styles. We also cover the wide range of proven data structure layouts already used in industry - each with its own capabilities, challenges and recommended usage domains.

We are particularly interested in where data structures and mathematics meet. For example, this course explores order theory and complexity theory and brings the richness of certain mathematical structures to common programming libraries (e.g. mathematical set has extra useful features that are not present in common implementations of sets in various runtimes). We also pay attention to correct usage of terminology to avoid misnaming – e.g. we see in C++ STL use of the term "vector" for something that is not a mathematical vector (an STL vector is a mathematical sequence [dynamic array]) - it is now too late to correct that misnaming.

Contents of One-Day Training Course

Target Audience

This course is aimed mathematicians and developers who wish to gain a richer understanding of modern approaches to data structures.

Prerequisites

Attendees need a good background in mathematics and programming.

Goals of Data Structures
Designing data structure the right way
Important considerations (future evolution, performance, compression, ..)
Accessibility (random vs. sequential)
Visibility (public vs. internal functionality)
Complexity and Big O notation

Common Data Structures
Review of common data structures from a mathematical viewpoint
Mathematical sequence = array
Mathematical set = set
Mathematical morphism = map
          (sometimes called dictionary)
How best to implement

Additional Data Structures
Linked list (how to get same performance from singly vs. doubly linked list)
FIFO queue (stack)
LIFO queue
Does each data structure need to be independently created, or can it be layered on top of another (adaptors)?

B-Tree
B-Tree is the most important data structure for storage architectures
Performance characteristics of B-Trees
Variations of B-Tree layout

Splay Tree
Introduction to binary search tree
A splay tree is a binary search tree
Recently used items quick to re-access
The splaying operation

Priority Queue
Priority for an item
Sorting on that

Hashing
Hashing plays a particular importance in many searching and sorting algorithms
Examine from first principles how efficient hashing works

Common Operations
Searching
Sorting
Merging
Splitting
Parallel access

Data Structures For Functional Programming
There are particular considerations to designing data structures for use with functional programming languages
Immutability and versioning

Algorithms
Structure independent algorithms
Applying algorithms in a uniform way
Use of iterators to traverse structures

Memory
For data structures that grow and contract over time, need good memory management
Memory management and allocators
Slab allocators

Project
Creating a custom graph engine to store and interact with large-scale graphs efficiently

ARM Assembly
Instruction Set, Dev Env, Memory, Registers, Context Switch, Control Flow, SIMD, Project

ARM is a high-performance/low-power consumption microprocessor architecture from ARM Holdings (owned by Softbank). It powers almost all of the world's mobile devices (Apple, Samsung, Google), many IoT devices and edge devices and is beginning to see deployment elsewhere, such as the data center.

Learning ARM assembly is more straightforward than its main x86 rival. ARM's simple yet powerful RISC instruction set has a more modern coherent overall design, compared to the CISC x86 instruction set which has evolved in a haphazard manner over decades.

It makes considerable sense for senior software developers creating system code and application code for ARM devices to know ARM assembly. Practical benefits include gaining maximum performance, debugging down to instruction set level, mitigating security threats, etc. and in general just knowing what is going on with code as it executes.

If you wish to rapidly get up to speed on ARM software programming at the foundational level, then this course is for you. It covers everything from setting up a dev environment to the instruction set to app design.

Contents of One-Day Training Course	
Target Audience System software developers seeking to get the most out of their applications running on ARM-based devices. **Prerequisites** Good all-round low-level programming experience. Knowledge of a low-level programming language such as C, C++ or Go. Previous knowledge of assembly programming useful but not essential.	**Assembly Features** Issues in modern processor design Assembly as a programming language CPU Instruction Set Architecture (ISA) Machine code, microcode & micro-ops **ARM CPU Architecture** High-level processor architecture family The ARM CPU's role as part of a modern computer system (bus, devices, I/O) Memory addressing ARM has a high register count ARM mode vs. Thumb mode **The ARM Instruction Set** Tour of the ARM instruction set Registers, cache, SIMD, float-point, etc. Traps and interrupts **Tools** Assembler, disassembler, profiler, debugger, dumper, other GNU toolset (e.g. binutils, gdb) Actual ARM system vs. QEMU Typical ARM developer workflow **Memory Management** Memory and the ARM CPU Detailed examination of registers Load and store (`adr, ldr, stm`) DMA, main memory, stack pointer Addressing **Strings, Integers, Floats** Representing numeric data types Basic mathematical calculations Strings and their operations Storing, loading and manipulating **Conditionals** Flow control opcodes, condition codes Labels, jumps, branches (e.g. `blt, beq`) **Functions** Calling functions (with parameters) Creating functions (prologue, epilogue) **The OS and System Calls** How system calls are exposed to assembly Calling the OS API / error handling **SIMD Programming** SIMD features / vectorizing designs SIMD programming using C intrinsics and assembler instructions **Threading And ARM** Reviewing of multithreading concepts Run to completion Synchronization Context switching **Security on ARM** How to mitigate exploits Security-relevant instructions Countering buffer overflow **Other Microprocessor Families** Contrast ARM with competitors (e.g. PowerPC, x86-64, RISC-V) Interesting futuristic architectures **Project** To wrap up, the attendees will be asked to complete a detailed project that brings together what they have learned in the course. The objectives will be to optimize processing performance, data throughput and developer productivity

x86-64 Assembly
Architecture, Opcodes, Primitives, Functions, Flow Control, OS Calls, C interop, Virt, SIMD, Project

"What Andy giveth, Bill taketh away". As the processor folks manage to speed up the computer, the software folks invariably manage to slow it down with layer upon layer of software inserted between the application and the CPU. It is a major problem that many of today's application developers have little or no appreciation of what is happening "under the hood" as their code mysteriously interacts through all these opaque layers with the CPU. The solution is for developers to learn assembly for the CPU their applications use. This intensive course provides a whirlwind tour of major features of 64-bit x86 assembly.

There are three real benefits to learning x86-64 assembly. Firstly, you can program the performance-critical sections of your app directly in it, and exploit rich features such as SIMD. Secondly, you can optimize code written in high-level languages (HLL). With more knowledge, you can make more informed decisions about structuring your HLL code. Thirdly, you can debug complex problems at the instruction set level. The x86 opcodes are what gets passed to the CPU to execute. In tough debugging scenarios the more you understand what is transpiring, the quicker you catch bugs.

	Contents of One-Day Training Course	
	Assembly Features	**The OS and System Calls**
	Common aspects of assembly	How system calls are exposed to assembly
	Assembly as a language	Calling the OS API
	Opcodes	Error handling
	CPU Instruction Set Architecture (ISA)	**C and Assembly**
Target Audience	**Microprocessor Architecture**	Passing data from one to the other
This course target experienced software developers who wish to learn about x86-64 assembly, so that they can code in it directly and so that they understand how their high-level languages actually execute, to help with debugging and optimization.	Machine code, microcode & micro-ops	How C code is accessible from assembly
	The CPU's role in a computer system	How assembly is accessible from C
	The bus, devices, I/O, addressing	Inline assembly
	The x86 Instruction Set	**Optimization**
	What makes up the instruction set?	Understanding the cost of each instruction
	Registers, cache, SIMD, float-point, etc.	Reducing the number of instructions
	Traps and interrupts	Using more appropriate instructions
	Application structure	**Stream Computing**
	Tools	Vector programming
	Assembler, disassembler, profiler, debugger, dumper, other	Single instruction, multiple data (SIMD)
		Opcodes for SIMD
	Integers And Floats	Highly efficient for certain workloads
	Representing scalar and floating numbers	**Virtualization**
	Basic mathematical calculations	Hypervisor
	Strings	Doing virtualization in software
	String operations	Modern on-chip virtualization
	Storing, loading and manipulating	Working with virtualization in assembly
Prerequisites	**Conditionals**	**Impact of 64-bit**
Good experience of software programming.	Flow control opcodes	Lengthening the registers
	Labels, jumps, branches	Address space changes
No previous assembly programming experience required.	**Functions**	**Other Microprocessor Families**
	Calling functions	Contrast x86-86 with competitors
	Creating functions	(e.g. PowerPC, ARM, RISC-V)
	Parameters	Interesting futuristic architectures
	Executable Formats	**Project**
	ELF (Linux) and PE/COFF (Windows)	Developing a high-performance custom
	How assembly is placed inside executable	solution in x86-64 assembly

WebAssembly And WASI
Embedders, Modules, WABT, Primitives, Functions, Linear Memory, Tables, Control Flow, JS API, WASI

All the major browser vendors - Google, Mozilla, Microsoft and Apple - have cooperated to define an agreed standard called WebAssembly that specifies how an executable looks like for embedders to execute. WebAssembly allows you to run code written directly in assembly or in a high-level language (e.g. C/C++) compiled into assembly in a browser without plug-ins.

WebAssembly is a virtual Instruction Set Architecture that runs in an embedder. Currently the most popular embedder is the modern standard web browser and on the server the latest Node.js 22 supports it. Other more specialist embedders include the OCaml based spec interpreter and the C++ based WABT interpreter and Fastly's Lucet. WebAssembly modules are binary, low-level (e.g. support 64-bit integers, unlike JavaScript), and are very fast both to load and run. Support for calling WebIDL- defined APIs (e.g. the DOM) will be added in future. WASI provides a system interface.

Web developers with requirements for very high performing applications will benefit from exploring WebAssembly as it the basis for considerable industry innovation and has wide cross-browser support.

	Contents of One-Day Training Course	
Target Audience Developers wishing to get the very best performance out of their web browser and web server code **Prerequisites** Knowledge of C/C++ /JavaScript/TypeScript with some background experience of working with assembly	**WebAssembly Overview** Part of the modern web platform Designed for the web (security, etc.) Tour of WebAssembly concepts **WebAssembly Modules** Binary files with .wasm suffix Can be written by hand (assembly programming) or compiled from C/C++ Overview of assembly syntax S-Expressions **Embedder** Role of embedders Embedding in web browsers Embedding on server (e.g. Node.js 22) Spec and WABT interpreters Custom embedders in your applications **WABT – WebAssembly Binary Toolkit** Low-level CLI tools to work with wasm modules Assembler, disassembler, interpreter, linker, extractor etc. **Primitive Types** Just four!! - `i32` / `i64` / `f32` / `f64` Working with primitives Primitive operations **Functions** Defining and calling functions Function parameters and return value The `call` opcode The `start` function Importing/exporting functions	**Linear Memory** Handling strings in linear memory Memory imports and exports Data section **Structured Control Flow** Hierarchical targets Don't jump to specific address, rather move up levels in hierarchy `block`/`loop`/`if`/`else`/`br_table` opcode **Tables** Indirect function calls and security Tables and the call_indirect opcode Table imports and exports **JavaScript API** Think of a WebAssembly module as a low-level representation that is passed to web browser for internal/local compilation Calling from JS to wasm / wasm to JS **Module Binary Format** Binary format with well-defined layout Sections are either named or indexed Extensible – can add custom sections LEB128, opcodes, .. **WebAssembly Threads** How threads and webAssembly interact Shared state via shared array buffer **Future Enhancements** Future enhancements coming in areas such as SIMD, exceptions, garbage collection and synchronization **WASI** The new WebAssembly System Interface carefully defines a portable system API

.NET Intermediate Language (IL)
Stack Engine, IL Fundamentals, Metadata, IL Syntax, Opcode Injection, Building Compilers, Reflection

Intermediate Language (IL) is .NET's low-level platform-independent representation of an executable. Many .NET developers are content to write high-level code in an IDE and then compile/run it, oblivious to IL. More advanced developers and those with specialist needs are more ambitious – they wish to program directly in IL, to browse and edit the IL generated for them by high-level language compilers, to auto-generate source code from other logical representations, to create compilers, and to really know "under the hood" how code runs when using high level languages (to help optimize performance, aid more precise debugging,etc.)

This course covers all aspects of IL, including the opcodes, metadata, assembly syntax, compilation/de-compilation tools, binary file format and .NET's reflection (which provides classes to browse existing assemblies and to emit assemblies directly). We also examine usage scenarios, such as building your own compiler and code generation tools.

Attending this course will allow you get a jumpstart on understanding all aspects of .NET IL, to produce a variety of code manipulation functionality and gain a much better appreciation of how .NET code executes.

	Contents of One-Day Training Course	
	Review of CLR Issues	**Other IL Features**
	Assemblies & modules, how code executes, security issues, type loader	Unmanaged code
	CLR architecture from IL viewpoint	Exception handling/events/delegates
	Stack-based execution engine	**Programming with IL**
Target Audience	**IL Fundamentals**	Writing more complex programs in IL
This course will interest advanced .NET developers who wish to code directly in IL, or who need a richer understanding of how their higher-level code executes, or who need to create code generators and specialist developer tools.	Overall IL Model	Coding issues to be aware of
	Verbose/compact IL	Object interactions in IL
	JIT compiler	**Profiler API**
	Hello world in IL	The unmanaged Profiler API allows you to add custom code that will be called when the CLR is about to JIT IL code
	IL Tools	
	Ilasm.exe, Ildasm.exe	You can change the IL on-the-fly
	Ngen.exe, PEVerify.exe	**Code Interactions**
	Introduction to Structure of IL	Coverage of why and how one might wish to programmatically interact with code
	PE/COFF headers and sections	Overview of required code services
	Metadata tables	**Reflection**
	Manifest	System.Reflection namespace
	Managed code representations	Dynamically loading & invoking types
	Metadata Fundamentals	Browsing contents of assemblies
	Set of tables with very detailed data about contained code; Table types and uses	**Emitting**
Prerequisites		System.Reflection.Emit.*-Builder classes
In-depth knowledge of C# 12 and all round experience using the .NET CLR	**Advanced Metadata**	Emitting persistent & transient assemblies
	Important tables (ModuleDef, TypeDef, MethodDef, FieldDef, AssemblyRef, ModuleRef, ClassLayout, NestedClass	**.NET Native and IL**
		.NET Native – concepts and toolchain
	Types, Fields and Methods	Converting from IL to native code
	The IL instruction set	**Building Custom IL Tools**
Experience of language design, compiler creation and low-level code manipulation useful	Use of IL language constructs	Coverage of why and how to programmatically interact with IL code
	How code from high-level .NET languages appears in IL	Overview of required code services
	Advanced Types	**Project**
	Signatures, visibility, inheritance, ctors	How to integrate IL modules in your own custom project
	Primitive/native/managed types	

Coq Proof Assistant
Overview, Setup, Inductive Types, Standard Lib, Tactics, Automation, Extraction, Project

Among future-thinking software engineers worldwide there is considerable interest in creating verified software based on rigorous mathematical specifications. Among modern mathematicians having to deal with increasingly complex constructive mathematics, the idea of using computers for verified proofs is gaining traction. Recent proof assistants (e.g. Coq) provide the necessary tooling for both, as they are variations on the same theme (propositions as types/proofs as programs).

Initially created by France's leading research institute, Inria, Coq is now the world's leading in-production proof assistant with a thriving open-source project with a vibrant ecosystem and excellent documentation (e.g. Pierce et al. and Chlipala).

Coq is based on the Calculus of Inductive Constructions and has attracted many innovative ideas as to how best represent proofs and their "execution" by computer.

The idea of proof assistants will play a central role in mathematics and software development in the 2020s. This intensive course brings mathematicians and developers up to speed on what they need to know.

	Contents of One-Day Training Course	
Target Audience Future-focused mathematicians and software developers who wish to explore mathematical proofs and verified programming as if they are the same **Prerequisites** Knowledge of functional programming in any functional language (as Coq can be considered an advanced form of a functional language).	**Coq Overview** What is a proof assistant and what does it actually do? How does a proof assistant work? Intro to Coq and other proof assistants Tour of Coq features **Coq Dev Setup** Tooling needed to work with Coq CoqIDE coqdoc make system Documentation resources **Intro to Coq Programming** The Gallina specification language Simple types (enumerations, booleans, natural numbers, ..) General layout of a Coq program Control flow Functions `Require Import` command **Induction** Creating a simple inductive type Using enumerations Pattern matching Proving properties **Logic** Easy to represent mathematical logic in Coq – implication, conjunction, disjunction, negation **Working with Combinations** Lists, sets and maps Finite variations of these Relations	**Tactics** "*The tactics implement backward reasoning. When applied to a goal, a tactic replaces this goal with the subgoals it generates. We say that a tactic reduces a goal to its subgoal(s).*" [link] **Automation** `auto` tactic for proof search – tries to solve the current goal The `Ltac` tactic language `eapply` – apply with existential variables `eauto` – a general auto tactic **Coq Standard Library** Coq has a comprehensive standard library with functionality related to logic, algebraic structures, bools, number types, relations, sets, strings, Unicode and lots more **Extraction** To execute code, do not need entire Coq specification, use simpler representation Hence extraction is the process of creating code in a functional language to run The `Extraction` command Multiple target languages (OCaml, Haskell) **Internals** Coq source written in OCaml How does Coq work under the hood Exploring how extraction works in detail **Project** This course concludes with a two-part project requiring: * using Coq to provide mathematics proofs * using Coq to write functional software

Fundamentals Of Object Programming
Modern OOD, Design By Contract, Reusability, AOP, MDA, Design Patterns, Software Failure, Generics

Object oriented techniques has now been applied to a large number of projects with a varying degree of success. With a better understanding of where the real benefits lie, object technology is being continuously refined and improved accordingly. This training course brings software developers - who already have a working knowledge of the basics of OO - up to speed with modern object-oriented design and the evolving OO programming techniques and enables them to discover how to use objects successfully. It answers the question: "What is happening in the OO world, beyond the fundamentals such as inheritance, encapsulation and polymorphism?" The goal of any OO project is to produce the best software for as few resources (time /money/devs) as possible. The software should evolve well in the future and adapt to changing needs. Parts of the software should be reusable on other projects. It should behave robustly when it encounters errors.

You will benefit from attending this course by gaining a clear understanding of the very latest object technology concepts, understanding its vocabulary and identifying how it fits into the broader picture of s/w engineering when describing how to approach software tasks.

	Contents of One-Day Training Course	
Target Audience Software engineers wishing to learn about the latest advances in object-oriented concepts **Prerequisites** Some previous OO software development and general application design experience.	**Object Modeling** Why do we need to model? What benefits does it produce? Object technology involves a series of simple concepts – why is it so difficult to really get right? **Design By Contract** Design By Contract concepts Preconditions/postconditions/invariants **Object Discovery** How do I find the appropriate objects Object discovery techniques Is this one object, two objects, or none? **Modern OOD** Getting the class and instance structure right. The importance of typing and ADT. Meyer's Software Construction Principles: Linguistic Modular Units Self-documentation Uniform Access Open-Closed Single Choice **Reusability** How to reuse objects Design for reuse Managing reuse **Aspect Oriented Programming** What are aspects Reuse through the callstack Setting up the call environment with AOP **MDA** Concept of Model Driven Architecture How to apply MDA	**Design Patterns** The essence of a good project is not in its codebase but in its design A good design is far more reusable than a good piece of source code Design patterns are a technique of capturing design experience for later reuse by the original and other designers **Software Failure** Identifying failure points Catching, reporting and reacting to failure Importance of handling failure within OO **Generics** Type-independent programming Generics in various languages **Agile Modeling** Problems with "heavy-weight" processes AM is a family of light-weight processes, suitable for demands of fluid projects Modeling the AM way **eXtreme Programming (XP)** XP Principles: pair programming, integrate & test every day, continuous feedback, get to running code ASAP, evolutionary design, managing communication **Latest OO Ideas** Orderly vs. Experimental Engineering AntiPatterns Internet time Designing for continuous change **Case Study** Detailed case study showing how to apply advanced OO concepts to a project

C 23
Data Types, Functions, Conditionals, Pointers, Functions, I/O, Structures, Pre-processor, DLLs

C is available on virtually every form of computing hardware and for every OS. C libraries are callable from almost every high-level programming language (this is needed because the API to most OSes is in C). Applications seeking close-to-the-metal performance need to be written in C. Most kernel and device driver development is in C. The syntax of custom languages for GPUs (shader languages) and FPGAs are C-like.

Hence every developer should know C and know it well. What you learn on this intensive training course will be applicable on all these target platforms.

In this course all the main C programming concepts are covered. This includes flow control constructs, pointers, functions, the pre-processor and typedefs along with the importance of data-types, type safety and custom structures. Compiling, linking and debugging multi-file applications are covered with demo code that evolves from simple ten-line utilities to large-scale projects involving multiple developers (e.g. libraries & header files). As well as covering the syntax of the C language, we also cover how to use the important C standard library (e.g. I/O) and explain how to use C effectively in app development.

Contents of One-Day Training Course

Target Audience

This training course is aimed at software engineers who need to quickly get up to speed developing in C.

Prerequisites

Attendees must have a programming background but no experience of C required.

Overview of C
C evolution – K&R, ANSI, C99, C11, C18
Whirlwind tour of features of C
Hardware issues (e.g. volatile)

A Simple Program
Freeform language
Data types and variables
Constants, functions, comments, layout

Variables and Data Types
Simple data types, naming and size
Strings - char and wchar
Security and buffer overflow issues
Enumerators
User defined structures and unions
Static and global variables
Visibility of global variables

Intro to C Functions
Splitting features into functions
Small and self-contained chunks of code
A multi-function C program

Operators
Arithmetic & relational operators
'=' and '= ='
Increment and decrement operators
Bitwise operators

Flow of control
if / if else / for loop
switch/case
while
break and continue

Non-Local Jumps
Practical uses of non-local jumps
setjmp and longjmp

Functions in Detail
Function prototypes
Variable arguments lists
Functions in separate files (.c and .h files)

Pointers and Arrays
Handling arrays and array arithmetic
Pointers to standard data & custom structs
Function pointers (defining/setting/calling)
void vs. void *, use of restricted modifier

Pre-processor
#include, #define, #ifdef and #endif
Concatenation/stringizing/varidic macros

Dynamic Memory
malloc, free, realloc; costs of memory ops
Examples of dynamic memory usage

C Standard Library
Standard I/O, file I/O
Assert, math, conversions, time

Compiling, Linking, Debugging
Role of modules and compilation units
Constructing apps and libraries from code in multiple files
Building, linking and debugging

C11 [major]/C18 [minor]
Tour of new features of latest standard
Anonymous structs
New C threads and stdatomics header files

Project
Object-based C - let's build OO in C with inheritance (tree of anonymous structs), encapsulation (redefining structs), exceptions (using setjmp/longjmp), namespaces, RTTI, generics, etc.

C++ 23
Classes, Inheritance, Visibility, Templates, RTTI, Exceptions, Namespaces, Class Hierarchy Design

This course provides developers with an intensive introduction to programming in C++, together with an overview of its powerful standard library. C++ is a general purpose programming language, highly suited for objected oriented and generic application, component and systems development. It is used for advanced, complex, full functionality projects. It is fast – and often selected where high performance coupled with a rich object-oriented language is needed.

Above and beyond the new syntax aspects of C++, this course also examines how to profit from a range of new object programming techniques that are evolving, such as genericity and refactoring. Also explored is the optimum architecture for modern C++ class hierarchies, the latest C++ standards (C++14, C++17, C++ 23) and interaction with C code. We specifically explore the interesting new C++ 23 features.

A core competency of all software engineers is to be having completely mastery of the programming language they use. If you are currently a programmer moving to a new C++ project, then you will greatly benefit from attending this course.

	Contents of One-Day Training Course
Target Audience This training course is aimed at people who need to quickly get up to speed developing in C++. **Prerequisites** Attendees should be experienced software developers with a good knowledge of any high-level programming language. No previous experience of C++ programming required, as this course covers the language from the fundamentals up.	**Tour of C++ Features** **Exceptions** Primitives Benefits, costs and recommendations Flow control associated with C++ exception handling Enumerations Families of exceptions Preprocessor Throw-try-catch syntax Compilation Declaring exceptions in header files **Classes** **Namespaces** Class members and this ptr Avoiding naming conflicts Visibility – public, protected, private The std and other namespaces Constructors and destructors Creating & using your own namespace static **RTTI** **Inheritance** Run Time Type Information Superclass and sub-classes Exploring objects + their types at runtime Constructors and the inheritance hierarchy **Refactoring** Impact of the virtual keyword Software grows and changes **Multiple Inheritance** Sub-optimal solutions evolve Benefits and problems Refactoring is a set of coding best Which function is executed practices which aim to fix & improve The diamond and vtbl issues **C Code Interaction** **Templates** How C++ code can call C code Function templates How C code can call C++ code Class templates extern C Function specialization **Design of C++ Class Libraries** How to write code once and then use it Classes are seldom needed on their own with a variety of data types How groups of classes can cooperate Debugging template code Designing hierarchies Designing with genericity in mind Optimally placing data and functionality **Lambda Expressions** **Advanced Language Features** What problem are these trying to solve Temporaries/ Header file changes Unnamed functions New keywords: mutable, typename, etc. Expression syntax Smart pointers:rationale & how to use Writing code that accepts lambda Potential problems to avoid expressions C++20: What's coming soon

The Java 22 Language
OO, Classes, Inheritance, Interfaces, Generics, Annotations, Exceptions, Events, Modules, JNI

With a much improved standards update cadence there is re-invigorated interest in advancing the Java language and ecosystem, by Oracle and external teams using and contributing to Java's evolution.

The Java 22 Language is a powerful object-oriented language with modern features such as the Java Platform Module System, lambdas, generics and annotations and a comprehensive set of OO features such as classes, inheritance, interfaces and exceptions.

Some of the largest enterprise projects in the world are written in Java. These are multi-year projects that are vital corporate assets and need to continuously evolve well into the future. Java is an important language for many university courses. Many high throughput cloud platforms are written in Java. The main language for Android is Java. Plenty of cutting-edge open source projects are written in Java. In addition to existing projects, Java is also being regularly selected as the main language for all kinds of new projects. Hence now is an ideal time to learn Java well – starting with the language itself.

	Contents of One-Day Training Course
Target Audience Software developers wishing to become Java developers, starting with learning the language itself. This is an ideal first course in Java. **Prerequisites** Programming experience with an OO language such as C++, TypeScript or C#, along with good exposure to object-oriented design. No previous experience of Java is needed, as this course covers the language from the fundamentals up.	**Java - Language & Ecosystem** Review of the ecosystem surrounding the Java language and how it is evolving Features - language, VM, bytecode, runtime services, framework, tooling What's new in Java 22 **A Java Project Walkthrough** Simple hello world project Main and command line args Primitive data types Code layout & basic syntax Classpath / JAR files **Classes And Constructors** Simple class definitions Static vs. instance members Access modifiers Variety of constructor layouts Nested classes **Inheritance** Inheritance trees Extending classes Overriding / hiding **Generics** Generic functions Different invocations Generic types (type variables) Shadowing Constraints **Annotations** We often wish to attach additional data to a class -without impacting the inheritance hierarchy Java annotations allow use of metadata **Interfaces & Mixins** Defining behaviors via interfaces Implementing multiple interfaces Uses for interfaces Default methods How to provision mixins using interfaces **Exceptions** Basic exception handling Creating an exception Throwing and re-throwing Catching an exception **Event Handling** java.lang.EventObject Observer and Observable Creating event listeners **Lambdas** Lambda expressions Quick and easy way to define a method Arguments and body Designing code using lambdas **Java Platform Module System** A named set of packages Module keyword Organizing modules **JNI - Calling C Code** JNI – Java Native Interface Calling out to C code from Java Passing parameters and accepting returning results between different languages **Java Project** Developing a Java project using a selection of language features showing how they can sensibly be used together

C# 12
C# Fundamentals, .NET Fx Intro, Types, Classes, Attributes, Properties (init-only), Generics, Streams

C# 12 is the premier development language for the .NET platform. C# was designed from scratch with .NET in mind. Most of the internals of .NET 8 and Visual Studio are written in it. It has been selected by the majority of application teams creating commercial .NET 8 projects. C# builds on the rich common heritage of languages such as C++ and Java - but avoids their pitfalls and adds certain interesting new concepts, such as LINQ. There are aspects of C# that developers already know, there are some they have experienced similar but slightly different syntax in other languages, and some that are innovative.

C# can be used to develop stand-alone apps, local and distributed components, web services and mobile code. It produces code that can target desktop PCs, mobile devices, servers and IoT devices. C# 12 can be used for DB [EF 6], UWP, ASP.NET, WebAssembly (Blazor) & security projects. Hence it is an excellent all-round development language for all .NET applications. This intensive course aims to take experienced software engineers rapidly through all the major aspects of C# 12 - using plenty of demo source code and hands-on labs to show it in action. This is an ideal first course for those moving to the C# 12 language and .NET 8.

	Contents of One-Day Training Course
Target Audience Experienced software engineers wishing to rapidly get up to speed with C#. **Prerequisites** Programming experience with an OO language such as C++, TypeScript or Java, along with good exposure to object-oriented design. No previous experience of C# or .NET is needed. This is an ideal first course for those wishing to start in the modern .NET ecosystem. This course covers C# 12 using Visual Studio.	**C# 12 and .NET 8** What is .NET? The Base Class Library The CLR How C# is used with .NET Delivery of C# functionality in assemblies **A C# Project Walk through** Solutions, projects and files Parts of a C# project Structure of code Setting up a solution with a C# app and class library **Base Types** Built-in data types .NET value types and reference types How C# and .NET data types compare Building code in C# that is callable from other languages **Language Fundamentals** Main starting point Flow control, operators Variables, methods Enumerators, bit flags, arrays, indexers Namespaces **Class Fundamentals** Members, constructors, visibility, ref and out, constant fields, structs Fields & properties, methods, nested types **Inheritance** Single inheritance only (for classes) Virtual functions / override / new with-expressions [C# 12] Designing libraries using inheritance **Delegates And Events** Equivalent of function pointers Defining and exposing delegates Registering an interest in a delegate Async info with events Design pattern for event handling **Interfaces** When to use interfaces Multiple inheritance & hierarchies Abstract classes vs. interfaces **Exception Handling** `Try .. catch … finally` Detecting and responding to exceptions Strategies for exception handling **Generics & Constraints** Generics (for methods and classes) Constraints / Partial types Anonymous methods Type inferencing **Expression Bodied Members** Succinct member definitions Methods, constructors, properties, indexers **Specialist Features** Null conditional operator Auto-property initializer / `nameof` **Calling C Code** Calling out to C code from C# Passing parameters / accepting return val **Records** A record is structured immutable data New data keyword **C# 12 - What's new** Review of the latest additions to the C# language and what benefits they deliver

TypeScript 5

Object Foundations, Classes, Mixins, Generics, Specialist Types, Iterators, Ambients, Lib.d.ts

For modern larger-scale applications that target the JavaScript VM, either in browsers (e.g. Angular 18) or on the server and command-line tools (Node), or mobile apps (Ionic) or desktop apps (Electron), many senior developers have a desire for a more robust and comprehensive programming language compared to JavaScript; and TypeScript is the answer.

TypeScript is a JavaScript-like language that transpiles to JavaScript so can run anywhere JavaScript runs. In addition to everything the JavaScript language offers, TypeScript also offers a much richer type system,

generics, decorators, interfaces, mixins, additional tools, ambient type declarations and lots more. This is convincing more and more project teams to adopt it as their core programming language. We see it use internally with Angular, Zone.js, RxJS, NgRx and many commercial applications (including very large ones).

The aim of this course is to quickly bring you up to speed with programming in TypeScript. We explore the language syntax, its access to libraries, how to build applications and see why it is more and more being selected instead of JavaScript by senior web developers.

	Contents of One-Day Training Course	

Target Audience
Developers wishing to create modern apps using TypeScript
 	
Prerequisites
Software developers with an object-oriented background and some browser programming experience.
	TypeScript Introduction
	Relationship to ECMAScript standards
	Language tour
	What we should be familiar with and what
	  may be new to us (any, never, tuple)
	tsconfig.json and transpiling
	Object Foundations
	Type system hierarchy
	Type inferencing
	Visibility and immutability
	Object Types
	Duck typing
	Type system
	ObjectType definition
	Properties and accessors
	Call signatures
	Tuples
	Classes, Interfaces & Mixins
	Defining a class
	Constructors
	Inheritance
	Specifying an interface
	What happens to interfaces after
	  transpilation (they disappear!)
	Partial or full implementation of interface
	Additional construct which can be very
	  useful in certain circumstances
	Generics and Constraints
	Type-independent code
	Separating algorithm from types
	Constraining permissible type parameters
	Relationship to transpiled code
		Namespace & Modules
		Modules as a unit of delivery and unit of
		  code management
		Importing and exporting
		Sub-dividing module types in namespaces
		Use in conjunction with module naming
		Iterators & Generators
		Symbol.iterator and for..of
		Generator function
		Specialist Types
		Intersection type
		Union type
		Nullable
		Alias
		Reflection/Decorators/Metadata
		Attaching metadata to a class
		Using decorators
		The reflect-metadata package
		Ambient Declarations
		Interacting with non-TypeScript libraries
		  and use of @types with npm
		Writing and publishing .d.ts files
		Ambient syntax
		lib.d.ts Standard Library
		A modular collection of ambient
		declarations for various targets
		New for TypeScript 5
		Exploring the new features of TypeScript 5
		– including variadic tuple types, labelled
		tuple elements, class property inf and more
		Project
		Using TypeScript in a project
		  to build a modern flexible framework

The Go Language
Goals, Workspaces, Go Type System, CGo, Packaging, Concurrency, Remote Repositories

The Go Language (GoLang) is a next-generation system programming language that is rapidly growing in popularity. Many wildly successful commercial and open-source projects such as Docker and Kubernetes are written in Go. It is appreciated for a range of innovations, such as built-in concurrency including its channel architecture, simplified tooling, sensible conventions, and lots more. This course is aimed at experienced developers and brings them quickly up to speed programming Go and being able to understand and enhance existing Go source trees and being able to build their own.

We cover creating Go commands and libraries, package management, interaction with repositories, structural typing. We look at Go's rich range of system packages. We see how lack of classes and inheritance is not a problem. We explore the use of Go in a range of popular projects and see what real-world benefits it brings to system-level programming.

Go is the language of choice for modern low-level work and it is increasingly being selected by cutting-edge system developers for their most challenging projects. This course helps each attendee become one.

	Contents of One-Day Training Course
Target Audience Developers wishing to create modern system-level applications using the Go language. **Prerequisites** Software developers with an object-oriented and system programming (e.g. concurrency) background. Knowledge of C or other system-level language is a plus. **Notes** All samples and labs in this course use Go 1.15 - the very latest production release. Our instructor will use the GoLand IDE; attendees may use any suitable editor they wish to select	**Go Introduction** What problem set is Go trying to solve? A sensible evolution of C Excellent for system programming Setting up Go on your dev machine **Tour Of Language Features** Goroutines Packaging Building Functions as first class citizens Interfaces and structural typing Lack of classes; no inheritance **Workspace Management** The Go workspace is how code is managed Naming and file placement conventions Handling packages Building & using commands and libraries Role of `$GOPATH` `go build` vs. `go install` Creating / importing packages / exports **Go Type System** bools, strings, ints, runes (codepoints), floats, complex Zero (default) values Conversions Type inferencing Variables with var, :=, constants **Go Functions** Defining function signatures Returning more than one result Named returns Advanced function usage	**Constructs** `if` and `if-else` statements `for` loop / `defer` statement `switch` (how break is different) ("C's `while` is spelled `for` in Go" !!) **Grouping of Data Items** Arrays and maps Slices Structs and pointers **Types and Their Interfaces** Methods – functions with a receiver arg Interfaces define sets of methods **Concurrency** Concurrency primitives built into Go itself goroutines and channels Synchronization **Testing** Go has a built in testing system Test preparation The testing package Executing tests with `go test` **CGo – Calling C Code** Most OS APIs are written in C Need to call them and other C libraries How the pseudo-package C works How Go and C code can interact **Remote Packages** How to access remote repositories Use of `go get` Incorporating remote repositories into app **Project** Review of Go usage in a larger project.

Python 3
Language Features, OO Constructs, Framework, Pip Installers, IPython, Jupyter Notebooks, Testing

Python is the world's most versatile language. Unlike other languages that tend to be really good at one area but not so good at other areas, Python is really good at many areas. We see Python being successfully used for AI and machine learning, general application development, rich shell scripting, configuration, build systems, interactive notebooks, and lots more. Some of the largest technology companies in the world (e.g. Google) heavily use Python for their engineering and production systems. Some of the latest hot technologies (e.g. Keras and TensorFlow 2) use it extensively; so now is time for your team to use Python.

This fast-paced course covers all important aspects of Python programming., It is aimed at multi-disciplinary software engineers already experienced with object oriented programming using other languages. They will find much of their hard-earned knowledge easily transfers to Python programming – albeit delivered via a significantly simpler and more compact syntax. Invariably a given algorithm written in a different language when re-written in Python will result in smaller amounts of code, which is an excellent result (after all, the best developer writes the least amount of code).

Contents of One-Day Training Course	
Target Audience Developers wishing to create modern apps using the very latest version of Python. **Prerequisites** Software developers with practical programming experience of an object-oriented languages such as C++, C#, or Java. No prior Python experience needed.	**Python Tour** What Python offers Feature tour What make Python different from competing languages Emphasis on clean syntax **Language Constructs** Common data types Control flow Loops Functions RegEx **OO Programming in Python** Classes: layout, methods and attributes The __init__() method Inheritance Typing **Runtime Features** Memory management Generators Modules Multithreading & locks (threading.py) **Error Handling** Raising and catching exceptions (try, raise, except, ..) Designing with error handling in mind **Framework** The Python Standard Library offers: * Collections * File I/O * Data access * Network programming * User interface **Pip** Standard installer Pip usage Python Packaging Index Virtual environments Python modules **(Interactive) IPython** Interactive shell that supports a wide range of Python features, from visualization to threading to data access Also useful for and other languages **Jupyter Notebook** Jupyter Notebook mixes code, execution results, visualizations and markdown content in a single deliverable **Embedding Python** Many apps could benefit from a built-in macro language and Python is optimum We explore how to easily embed Python runtime in your custom application **Testing** Exploring Python's testing infrastructure Unit testing – what's similar & different Mocking Debugging **Interacting with C** Most OS APIs are written in C How Python and C code can interact – threading, memory, lifecycles, exceptions Handling common data types & constructs **Project** Using Python in a larger project to highlight its real-world capabilities

OCaml Functional Programming
FP Concepts, OCaml Language, OCaml Tools, OCaml Library, Dune, Opam, Projects Using OCaml

" OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles " [https://ocaml.org]. OCaml is best known as a functional programming language and that is what we focus on in this intensive course. OCaml competes with Haskell to be the leading functional language. For a number of reasons, we prefer OCaml. It has a number of advanced features, a richer type system and a more extensive system library. It is also used on many cutting edge projects that interest us. Examples of practical uses of OCaml include: the experimental redtt (based on cubical type theory) and the well established

Coq proof assistants, samples in the important TAPL book, the WebAssembly spec interpreter, the mirageOS unikernel and Jane Street.

Functional programming is different from regular object-oriented programming. So we start by looking at FP for non-FP programmers. Then we explore all aspects of programming with OCaml – the language, tooling and system library. We also explore add-on libraries. Our goal is to ensure all attendees are up to speed with OCaml programming and immediately after this course can be productive as OCaml developers.

	Contents of One-Day Training Course	
	FP For non FP Programmers	**OCaml Library**
	What is functional programming?	Structure of standard library
	Mathematics is important for modern code	Tour of main functionality areas
	What is a variable (immutability)	Common collections
	Pattern matching	Text handling
Target Audience	Handling change	Threads library (modules: thread, mutex, condition, event, ..)
Experienced software developers who wish to start functional programming using the OCaml language.	Pure functions	
	Building more reliable code	Async and deferred computation
	Many FP ideas are seeping into other types of programming – why and how?	**Tooling**
	OCaml Tour	Debugging
	Language basics (control flow, etc.)	Testing
	Structure source trees and individual units	Lint
	`main` function-conventional/not necessary	Pre-processor
	Creating functions	Compilation tools – (parsing, etc.)
	Events and callbacks	Abstract syntax tree
Prerequisites	Type inferencing	**Dune**
Attendees need prior programming experience in one of the mainstream object-oriented programming languages.	**OCaml Language**	Dune is OCaml's composable build system
	Modules (submodules)	Defining steps needed for build
	.mli interface definition files	Configuration
	More detailed look at functions	S-Expression syntax
	Visibility	Compilation flags
	Objects and object types	**Opam**
Important: This course assumes attendees have no prior functional programming experience, so in addition to teaching OCaml, we also cover general aspects of functional programming.	Classes / polymorphism / class types	Opam is the OCaml package manager
	Type mismatch and other errors	Package repository
	OCaml Runtime	Extensive collection of pre-built packages
	OCaml runtime architecture	Managing locally installed packages
	How code executes (e.g. as native code)	Packaging definition file – creating and populating for a custom project
	Dynamic linking -what's involved in dynamically loading a library	**Projects**
	The GC	Exploring usage of OCaml in a variety of open source projects that have shipped, to see how everything fits together in a larger production setting
	Foreign function interface - interacting with C code (and other languages)	

The Bash Shell
Tour, Variables, Control Flow, Functions, Builtins, History, Error Handling, Aliases, Best Practices

Bash is a powerful shell environment that provides excellent interactive and scripting control of the underlying OS, installed software platforms and can be used to build and manage your own custom tools too. Bash is good for repetitive chores, setting up an environment, automating non-trivial workflows, software builds and product installations.

The goal of well-written shell scripts is to allow repeatable / configurable / reliable task execution. The overall aim of this course is to equip attendees with a clear understanding of how to achieve that using Bash.

Professionals skilled in the art of Bash scripting are significantly more productive. Knowledge of Bash is a "must-have" tool in the skillset of every heterogeneous system administrator or developer. Though initially popular on Unix and Linux platforms, Bash is now very widely available, including on macOS and Microsoft itself has added it to modern Windows (11 & Server 2022, as an optional install). With a little effort you can even get Bash to work on mobile OSes. This wide availability is important because it means the effort you invest to learn Bash well now will pay repeated dividends in future regardless of which OS you use.

Contents of One-Day Training Course		
	Overview	**Scripts**
	What is Bash good for?	Creating/calling/debugging Bash scripts
	Compliance with POSIX Shell Standard	Sourcing – external library of functions
	Feature tour	Job control
	Setting up Bash – for Linux/macOS,	History
Target Audience	Windows Subsystem For Linux and	Arranging larger blocks of script
Administrators, developers	Git Bash (limited)	**Builtin Functions**
and power users who wish	How Bash compares to PowerShell	Wide range of builtins
to use the command line to	**Control Flow**	Regular expressions
interact with the system	if	readline
and to create shell scripts	select / case	**Alias**
to automate such activities	for	Listing aliases
	break / continue	Role of aliases
	Variables	alias and unalias functions
	set and unset for local variables	**Error handling**
	env	Error handling in scripts
Prerequisites	Export to child processes	Signal handling
Some previous experience	**Processes and Command Line**	trap
of shell usage required.	Command pipeline	**File Handling**
	Command line arguments	How files are represented in Bash
General knowledge of the	Exit code and results	general file I/O
operating system	Process architecture of executing Bash	umask
command layout is useful.	exec command	read
	$$ - process id of process executing shell	**Scripting Best Practices**
	Functions	Small chunks of script
	Parameters to functions	Pay attention to debugging
	Viewing declared functions	Resilience to error situations
	Nested functions	Similar to/different from regular
	Job control	programming
	Configuring the Environment	**Project**
	.bashrc	Explores the scripting needed for
	Start up scripts - profiles	automated workflow for a non-trivial
	Login shell	enterprise scenario

PowerShell 7
Setup, Syntax, CLI, CmdLets, Control Flow, DSC, Modules, Security, Remoting, .NET Integration

PowerShell 7 is Microsoft's latest innovative scripting and automation engine. The older versions of PowerShell were based on the .NET Framework and only ran on Windows. In contrast, the new PowerShell is based on .NET and and so can run on Windows, Linux and macOS. Both PowerShell and .NET are open source. PowerShell is the basis for Azure Cloud Shell. PowerShell commands have a very flexible syntax and can be executed immediately or stored in a script, and later executed on the local machine or (after appropriate security steps) on remote machines.

PowerShell commands are best created via the PowerShell extension to Visual Studio Code; they can also be created via any text editor.

PowerShell has a few key concepts that separate it from previous shell languages. It is based on .NET and its syntax borrows from C#, so moving between both is quite easy. Like all shells, PowerShell works on the basis of a pipeline. Unlike other shells, what flows along the PowerShell pipeline are .NET objects. PowerShell offers the idea of consistently named cmdlets (<verb>-<noun>: e.g. `Get-Process` lists processes).

	Contents of One-Day Training Course	
	Tour of PowerShell Core Ecosystem	**Server Automation**
	Overview of all parts of PowerShell	PowerShell works with many kinds of server automation packages
	Installing and configuring	DSC Core and DSC Resource Kit
	Choosing an editor	Chef, Ansible, Puppet
Target Audience	Getting started with your first script	Scripting in Azure Cloud Shell
System administrators and software developers wishing to benefit from the modern approach to scripting on Windows, Linux and macOS.	Exploring main language features	**Security**
	CmdLets	Use of SSH as protocol for PSRP
	CmdLets (Command lets, as in, small commands) are individual pieces of functionality that can be used standalone or more usually combined to form larger scripts – we explore how they work	Signing, execution policy, authenticode
		PowerShell And OpenAPI
		Review of use of OpenAPI as a very popular definition syntax for precisely describing REST API
	Best practices for creating CmdLets	Using AutoRest to generate CmdLet client
	Variables	**Remoting**
	Think of PowerShell as a simple program-ming language, so variables needed	WSMan
	Scope. naming, usage	Windows management Framework
	More Language Constructs	PowerShell Remoting Protocol: PSRP
Prerequisites	Control flow	**.NET Extensions**
Good knowledge of scripting with any similar environment (e.g. Bash).	Functions	Using PowerShell to call a standard .NET assembly and also calling into third party .NET assemblies
	Exception handling with try / catch	Designing and building our own custom .NET assemblies that can be used from PowerShell
	Error info within exceptions using $_	
	Shell Constructs	
	Formatting output from within script	Architectural guidance for how to structure larger integration
	Accepting command line arguments	**Project**
	Environment	Many larger commercial server products are now being delivered with a PowerShell interface (e.g. Vmware's PowerCLI). We explore how to design something similar for our own sample server product
	OS interaction	
	Providers	
	To extend the range of data stores exposed to PowerShell scripts, you can create and load custom providers	
	Architecture of a provider+sample impl	

Technology / Compute School

OS

- Windows System Programming Using C/C++
- Windows Multithreading Using C/C++
- Designing Server Platforms for Windows Server 2022 Using C/C++
- POSIX And Linux 5 System Programming Using C/C++
- pthreads – POSIX and Linux 5 Multithreading using C/C++
- Designing Server Platforms for POSIX And Linux 5 Using C/C++
- Microservices and Containers

Runtime

- Node.js 22 Runtime Programming Using TypeScript
- Browser Runtime Programming Using TypeScript
- Async, Parallel And Reactive (RxJS) Programming Using TypeScript
- .NET 8 CLR Programming
- .NET 8 Multithreaded And Parallel Programming
- .NET Managed Extensibility Framework (MEF)
- ML.NET
- Java 22 Runtime Programming
- Java 22 Multithreading

Windows System Programming Using C/C++
Using low-level Windows OS API for maximum performance, security, extensibility, flexibility

This course examines how to use the Windows C API to design and develop advanced systems-level software. The Windows C API has matured and gained a rock-solid quality reputation with modern features such as threads, symmetric multi-processing, system-wide object model, powerful networking, asynchronous I/O and Unicode. The Windows C API is the common programmatic interface shared by all implementations of the Windows OS family. There are some differences in how it behaves on each OS, but it's possible to create a single EXE/DLL to run on all OSes. The focus of this course is the API of Windows [11 | Server 2022].

Important features in the areas of the registry, file systems, security and auditing are discussed. We cover the many techniques available for inter-process communication (e.g. pipes, mailslots, RPC and WinSock). Applications may be made run-time extensible by the configurable loading of DLLs.

Developers seeking extra performance and more flexible low-level control over OS system calls will benefit from writing their system-level application code in C/C++ and this course tells them what they need to know to quickly become productive.

	Contents of One-Day Training Course	
Target Audience System architects and experienced developers who need to create advanced systems using the Windows C API.	**The Windows Platform** Windows Architecture Windows SDK Issues of interest to app system developers **Overview of Windows C API** Applicability of Windows C API Important OS functionality Hardware issues Win32/Win64 on various OSes How to determine the underlying OS "Write to the API, not the OS" **General Architecture** Major OS components and subsystems Layout of OS and apps on file system Common Windows C datatypes & headers An application developer's view of the OS	**Registry** Complete coverage of the Registry APIs Handling configuration data **SEH** Structured exception handling Exception handlers Termination handlers **DLLs** Comparison of DLLs & EXEs DLL functions / variables / memory Explicit loading of DLLs using: LoadLibrary FreeLibrary GetProcAddress **C Run-Time Library (CRT)** Using the CRT with Win32/Win64 How the CRT is layered above the OS API

Prerequisites
 Attendees must have some previous experience of system-level programming
	Memory Management
	Flat memory structure
	Various types of alloc APIs
	Sharing memory / private heaps
	File System
	File APIs
	Directory handling
	Memory mapped files
	Security & Auditing
	How to programmatically interact with:
	    Security descriptors
	    Security attributes
	    SIDs and tokens
	    ACLs and ACEs
	    Privileges
	    WinStations
	    Desktops
		Inter-Process Communication
		Anonymous vs. Named pipes
		IPC using pipes & mailslots
		Distributing functionality using RPC
		Windows Networking APIs
		How to programmatically talk to the net
		Available APIs and when to use which
		Managing network connections with WNet
		WinSock & WinInet
		A protocol-independent API
		Relationship to Berkeley sockets
		WSA functions
		Coding client and server applications
		Blocking and non-blocking comms
		Advanced WinSock
		Socket options /out-of-band data

Windows Multithreading Using C/C++
Threading Concepts, Kernel Obj, Processes/Threads, Synchronization, Async I/O, Debugging, DLLs

[Sample: lab exercises] This course explores how to use the Windows C API to build sophisticated multithreaded architectures. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. The Windows kernel object is the opaque foundation for multithreading – based on this are the process, thread, job & various synchronization objects - mutex, event, semaphore and waitable timer – each of which targets different needs. Thread activity, lifetimes and inter-thread communication must be co-ordinated. Threads impact how to develop DLLs, memory and debug.

Various higher-level design patterns may be used to route workitems in multithreaded servers. The optimal server architecture is one active application thread per processor logical core. Tools may be developed to determine which thread is blocked waiting on which resource, and the state/owner of each resource. A server must efficiently multiplex many I/O requests over a few threads - which is the goal of I/O completion ports.

This course supplies attendees with an understanding of the concepts underlying threading, together with experience of multithreaded development.

	Contents of One-Day Training Course	
Target Audience System architects and experienced developers who need to gain an in-depth understanding of Windows multithreading. **Prerequisites** Attendees must have good experience of system-level programming, on either Windows or Linux.	**Thread definition** Scheduling vs. synchronization Parallelism and concurrency Compute-bound and I/O bound apps Race conditions, deadlock, starvation, priority inversion **Kernel Objects** Windows kernel objects Usage counting Kernel object handles Sharing handles among processes **Processes** CreateProcess API & child processes Retrieving the exit code of a process Job objects **Threads** CreateThread API & Threadproc The C runtime library Thread priority & processor affinity Thread management & lifetime **Synchronization** Critical Sections, Mutexes, Events, Semaphores, Waitable Timers, WaitForSingle/MultipleObject The "Protect data, not code" principle **Memory and Threads** Dynamic & static Thread Local Storage Heap storage vs. stack storage **Asynchronous I/O** Overlapped, APC & Scatter/Gather I/O Completion Ports, Asynchronous I/O, Overlapped, APC & Scatter/Gather I/O Completion Ports	**Thread Pools** OS-managed pools of threads for processing timers, work-items and I/O **DLLs and Threads** How threads interact with DLLs Serialized DllMain, shared sections Robust DLL design for threads **Debugging with Threads** Querying information about running processes/threads and their attributes The serialized OutputDebugString API **Resource Management** Creating a custom resource browser, to display which thread is waiting on which synchronization resource Threads with C++ Threads & exceptions; threads & classes Accessing resources using smart pointers **Design Issues** Single Writer/Multiple Readers, Monitor, Once-Off Initialization,Dining Philosopher Calling legacy code from multiple threads Converting legacy code to multithreading **Multithreaded Architectures** Pipeline, Producer-Consumer, Work-Crew Master-Slave Models Create threads on demand vs. elastic pool **Multithreaded Project** Development of a complete multithreaded embedded HTTP web server that uses I/O completion ports to efficiently manage large numbers of requests

Designing Server Platforms for
Windows Server 2022 Using C/C++
Installer, Service Process, Pipelines, Networking, Config, PerfMon, WMI, EIF, Patterns, HA/HT, Project

Server platforms consist of a mixture of multiple processes and threads, working in a co-ordinated manner, to provide some service to numerous clients on remote machines. These platforms must be flexible, extensible, configurable, scaleable and controllable. A pipeline architecture allows extensible processing of messages. Many techniques are available for flexible inter-process communication. Service processes are the best way to deliver long-lived non-GUI functionality.

Building server platforms for Windows Server 2022 is the logical choice for future-oriented projects.

If your team consists of senior developers experienced with Windows and C/C++ and your team is assigned the task of developing a high-quality server platform on Windows Server 2022, then this is the ideal course to get all team members up to speed on what is needed.

It covers design concepts, important Windows C APIs, plenty of code samples and a chance to have architectural questions answered. It explores extra features (such as ETW, clean installer, PerfMon) that will distinguish your team's platform from the competition.

	Contents of One-Day Training Course	
Target Audience System architects and senior software engineers who need to create advanced server platforms for Windows Server 2022 using C/C++. **Prerequisites** General Windows system programming and especially multithreading experience.	**Designing Server Platforms** Multiple Processes/Threads Central Service (Manager) / Worker Processes / GUI+CLI Admin Processes Variety of threading architectures **Service Process** SCM-service code interaction Install+config of service "Log on as a service" security right Controlling worker processes **Platform Installation** How best to install server apps Installer formats and server extensions **Platform Configuration** Rich config choices Web garden/web-farm layout Config changes without restarting **Pipelines** Processing paths for messages Sequential and non-sequential steps Structure of pipeline (handlers&modules) Pipeline context **Dynamically loading DLLs** Dynamically loading DLLs Updating a server's DLLs without having to restart it **Networking** High performance sockets design Eliminating buffer copying Async I/O & Scatter/Gather I/O **Use of Http Server API** Same http.sys kernel service as used by IIS itself; Advanced HTTP protocol	**Performance Monitor** Detecting bottlenecks Tuning performance Developing for Performance Counter **WMI** Windows Management Instrumentation Management classes Exposing your server through WMI Management by GUI and CLI **Event Tracing For Windows (ETW)** Tracing architecture Event tracing APIs **Designing Platform Security** Leveraging Windows' security features Defense in depth platform security Secure communication with remote clients **Design Patterns for Server Platforms** Patterns to satisfy competing demands Pooling, tuning, managing Sharing, distributing Extending, scheduling **High Availability / High Throughput** Hardware for high-availability/throughput ccNUMA, Interconnects SAN, DAS, NAS Clustering concepts Design for high availability/throughput **Project** Overview of development of a sample server platform for Windows Server 2022

POSIX And Linux 5 System Programming Using C/C++
Using low-level C API for maximum performance, security, extensibility, flexibility, portability

This course examines how to use POSIX standard APIs and Linux libc-specific APIs to create system-level software. POSIX defines specifications (documents) and operating systems (such as Linux with its libc library) or layered libraries (such as musl) implement these specifications (code). POSIX is supported on a variety of modern operating systems and so system-level application code should strive to use it as much as possible. Implementations can (and almost always do) add extra functions to the API list defined by POSIX and these extra functions provide very useful additional capabilities, though they limit portability.

Developers seeking extra performance and flexible low-level control over OS system calls will benefit from writing their system-level application code in C/C++ (rather than a language that comes with a heavy runtime, such as Java or C#). This course tells them what they need to know to become quickly productive.

We explore the entire path from application code to libc calls, through system calls right to the kernel where the actual functionality is delivered. We look closely at the architecture of how the OS programming interface is exposed to applications.

	Contents of One-Day Training Course	
Target Audience System architects and experienced developers who need to create advanced system software using POSIX and Linux libc APIs. **Prerequisites** Attendees must have some previous experience of system-level programming. Please note: This course does not cover pthreads (multithreading with POSIX) – we offer a separate detailed course for pthreads development.	**Big Picture** How everything fits together - POSIX, the standard C library, libc, Linux System Calls, the Linux Kernel Alternative implementations: musl, newlib Portability & POSIX on non-linux OSes Role of the Linux Standard Base (LSB) What's needed for development **Overview of POSIX Specs** POSIX.1-2017 : latest spec with sections on base definitions, system interfaces (APIs), shell & utilities, rationale "Write to the API, not the OS" Relation between POSIX and C ("POSIX is a superset of the standard C library" and libc API is a superset of POSIX) What is specifically outside POSIX scope **Linux General Architecture** Structure of Linux kernel and userspace Role APIs and system calls play Major OS components and subsystems Layout of OS and apps on file system **Linux libc** Common datatypes & headers An application developer's view of the OS Let's trace a system call from app to API to System Call Interface to kernel Trap 0x80 for x86 **Major POSIX Functional Areas** STREAMS, I/O, signals Regular expressions File descriptors & advanced file handling Process creation (fork/exec) & status	**Inter-Process Communication** IPC overview Message passing semaphores Shared memory **What libc Adds To POSIX** What's in libc above and beyond POSIX e.g. cgroups (underpins containers) Multimedia Newer libc system APIs not in POSIX Security concepts **Linux Binaries** File structure Programmatic interaction Creating binaries Dynamic loading of .so **Scheduling** Overview of Linux scheduling How apps use APIs to influence scheduling Priority **Sockets** How to programmatically talk to the net Available APIs and when to use which A protocol-independent API Coding client and server applications Blocking and non-blocking comms Socket options / out-of-band data Addressing, queuing signaling, errors, Advanced sockets **Large Codebases** Large codebases need more than POSIX Retain portability via e,g. Apache Portable Runtime or ACE

pthreads: POSIX And Linux 5 Multithreading using C/C++
Threading Concepts, Creating/Managing Threads, Synchronization, pthreads & .so, NPTL Internals

[Sample: concept map] This course explores how to use the pthreads C API to build sophisticated multithreaded architectures for modern POSIX-compatible OSes such as Linux 5 (e.g. Ubuntu 20.04 LTS). When designed correctly, multithreading can substantially increase app performance and responsiveness to distributed clients and end-users. POSIX defines a multithreading specification commonly known as pthreads. This is a C API that strictly specifies the expected behavior of threading and synchronization primitives. Code written to work against pthreads can run on any OS that implements this spec. Linux is one such OS and the focus for this course, but it is noted that pthreads is also implemented on a wide variety of other popular and specialist OSes. This course supplies attendees with an understanding of the concepts underlying multithreading, together with hands-on experience of multithreaded development on Linux. Topics covered include a comprehensive tour of thread creation and lifetime management, the various synchronization approaches, how threads interact with share libraries, memory access and debugging, intra-thread comms and various higher-level design patterns to work with large multithreaded servers.

	Contents of One-Day Training Course	
Target Audience System architects and experienced developers who need to gain an in-depth understanding of POSIX and Linux multithreading. **Prerequisites** Attendees must have good experience of system-level programming on Linux.	**Thread definition** Scheduling vs. synchronization Parallelism and concurrency Compute-bound and I/O bound apps Race conditions, deadlock, starvation, priority inversion **Threads** Tour of pthreads.h Overview of main APIs and C structures Creating a thread with pthread_create() The threadproc Thread priority Thread management & lifetime pthread_exit() and joining a thread to catch exit and to access exit code pthread_[detach|attr_setdetachstate]() Thread cancelation C11/18 threads vs. pthreads (quite similar) **Synchronization** Conditionals, mutexes, rwlock, spin, barrier – compare & contrast The "protect data, not code" principle pthread_cond_[init|destroy|attr_init]() pthread_mutex_[lock|trylock|unlock] Blocking vs. non-blocking pthread_[rwlock|spin|barrier]_init() **Thread Pools** Managing pools of threads for processing timers, work-items and I/O **Memory and Threads** Thread local storage : pthread_key_create Heap storage vs. stack storage pthread_attr_[set|get]stack[size|addr]()	**Shared Libraries and Threads** How threads interact with shared libraries Serialized methods Robust .so design for threads **Debugging with Threads** Querying information about running processes/threads and their attributes Serialized calls **Resource Management** Creating a custom resource browser, to display which thread is waiting on which synchronization resource Threads with C++ Threads & exceptions; threads & classes Accessing resources using smart pointers **Design Issues** Single Writer/Multiple Readers, Monitor, Once-Off Initialization,Dining Philosopher Calling legacy code from multiple threads Converting legacy code to multithreading **Multithreaded Architectures** Pipeline, Producer-Consumer, Work-Crew and Master-Slave Models Create threads on demand vs. elastic pool **NPTL Internals** Native POSIX Thread Library (NPTL) implements pthreads on Linux Threads and Linux scheduling **Multithreaded Project** Development of a complete multithreaded embedded HTTP web server that efficiently manage large numbers of requests

Designing Server Platforms
for POSIX And Linux 5 Using C/C++
Installer, Daemons/Worker Process, Pipelines, Networking, Config, Performance, Syslog, Systemd, Patterns, HA/HT, Project

Server platforms consist of a mixture of multiple processes and threads, working in a co-ordinated manner, to provide some service to numerous clients on remote machines. These platforms must be flexible, extensible, configurable, scaleable and controllable. A pipeline architecture allows extensible processing of messages. Many techniques are available for flexible inter-process communication. Service processes are the best way to deliver long-lived non-GUI functionality.

Building server platforms for Linux is the logical choice for future-oriented projects.

If your team consists of senior developers experienced with Linux and C/C++ and you are tasked with developing a high-quality server platform on Linux, then this is the ideal course to get all team members up to speed on what is needed. We recommend developing as much as possible using POSIX APIs (for portability), with careful use of additional APIs where it makes sense. This course covers design concepts, important POSIX APIs, plenty of code samples and a chance to have architectural questions answered. It explores extra features (such as Systemd and syslog) that will distinguish your team's platform from the competition.

Contents of One-Day Training Course		
Target Audience System architects and senior software engineers who need to create advanced server platforms for Linux using C/C++. **Prerequisites** General Linux system programming and especially multithreading experience.	**Designing Server Platforms** Multiple Processes/Threads Central Service (Manager) / Worker Processes / GUI+CLI Admin Processes Variety of threading architectures **Daemon Process** Overview of systemd Install+config of daemon Security right Controlling worker processes **Platform Installation** How best to install server apps Installer formats and server extensions **Platform Configuration** Rich config choices (role of /etc) Web garden/web-farm layout Config changes without restarting **Pipelines** Processing paths for messages Sequential and non-sequential steps Structure of pipeline (handlers&modules) Pipeline context **Dynamically loading .so** Dynamically loading shared libraries Updating a server's shared libraries without having to restart it **Networking** High performance sockets design Eliminating buffer copying Async I/O **Use of Apache Server API** Building custom Apache HTTP Sever module to link it to your daemon process	**Performance** Detecting bottlenecks Tuning performance Developing with performance in mind **Monitoring** Building server platform with sysadmins in mind – what capabilites do they need? How to adding monitoring to your platform **Syslog** Logging architecture Event tracing APIs **Designing Platform Security** Leveraging Linux security features Defense in depth platform security Secure communication with remote clients **Design Patterns for Server Platforms** Patterns to satisfy competing demands Pooling, tuning, managing Sharing, distributing Extending, scheduling **High Availability / High Throughput** Hardware for high-availability/throughput ccNUMA, Interconnects SAN, DAS, NAS Clustering concepts Design for high availability/throughput **Project** Overview of development of part of a sample large server platform for Linux Focus as much as possible on POSIX APIs Portability and use of OS-specific APIs

Containers And Microservices
Concepts, OCI, runC, CNCF, Containerd, Docker, Microservices, App Architecture, Networking, Project

Microservices have revolutionized server-side application development. Most modern engineering teams have evolved to running as much code as possible in containers and benefit from the range of enhancements they offer.

As an application developer, you can think of a container as an efficient sandbox within which your process runs (multiple processes can run in the one container, but usually it is one process per container). Containers offer a sandbox based on unique namespaces and c(ontrol) groups (e.g. resource limits/accounting).

The world of containers is undergoing rapid transformation (Docker and its components - e.g. runc - are important but they certainly are not the whole story). Developers really need to understand how all the moving parts fit together in the modern container world. They also need to be aware of how similar and contrasting containers are with traditional "process in OS" approach. This course focuses on individual containers and how attendees can build microservices (slice of their application) to run inside-a recommended follow-on course on *Kubernetes* explores how to orchestrate clusters of containers in innovative ways.

	Contents of One-Day Training Course	
Target Audience Server application developers wishing to compose applications from many microservices running in containers. **Prerequisites** Developers experienced in server-side software. Knowledge of the Go language is useful. **Note** Please note this course does not cover Kubernetes (apart from a brief introduction). We have a separate course dedicated to *Kubernetes*, which we recommend attendees takes after this course.	**Container Landscape** What containers offers over VM approach Intro to microservice development Big picture review of the sometimes confusing container landscape: its standard bodies, specs, commercial businesses, cloud offerings, tooling **OCI -Open Container Initiative** https://www.opencontainers.org and https://github.com/opencontainers "An open governance structure for .. creating open industry standards around container formats and runtime" Three specs: * Image is for file system layout * Runtime is how to run a container * Distribution (new) is how images travel Open source projects – runc (includes libcontainer), image-tools, runtime-tools (runc is the runtime used by Docker and by most Kubernetes installations) **CNCF - Cloud-Native Computing Foundation** https://www.cncf.io "builds sustainable ecosystems .. around a constellation of high-quality projects that orchestrate containers as part of a microservices architecture." Review of CNCF projects, including ... * Containerd - https://containerd.io/ **Docker** The Docker toolsuite is the market-leading container platform – let's explore what	services it offers and how to use them Available for Linux, macOS and Windows **Kata Containers** OCI-compliant open-source project to run containers via a light-weight hypervisor (combine containers and VM approaches) **Building Microservices** Sub-dividing a large app into microservices As an application developer, what steps you need to take to prepare your code to run inside a container **Application Architecture** What steps are needed to build a microservice and how to optimize? Handling data, IPC, config, lifecycle, etc. **Microservices And Security** Under what security context does your microservice code run? Importance of isolation **Microservices And Networking** * CNI (container networking) - https://github.com/containernetworking * Envoy (distributed proxy) - https://www.envoyproxy.io **Tour of Source Trees** Understanding internals is good: exploring runc CLI and libcontainer (written in Go), containerd (written in Go), Docker (written in Go), Kata (yup, also written in Go) **Project** How best to partition a large server app to run as microservices in many containers Practical architectural guidance

Node.js 22 Runtime Programming Using TypeScript
Feature Tour, Event Loop, Non-Blocking IO, VM, Crypto, Utilities, Streaming, N-API, Project

Node.js is a cross-platform easy-to-use runtime based on Google's high performance V8 JavaScript execution engine (as found in Chrome). Node also comes with a well crafted and substantial framework which covers many application areas.

Node has a number of desirable core characteristics – simplicity (it is very easy to get started using Node); modularity (everything is based on modules, which can grow over time); extensible (both by JavaScript / TypeScript code and by C/C++ code [using N-API]) and asynchronous / event-driven (non-blocking IO).

This course cover the latest edition – Node.js 22 – from the ground up for developers with little or no previous Node experience but a strong desire to rapidly become proficient in Node. We compare Node to other popular runtimes that they might know and see there are many similarities (language VM, package management, framework layout, tools, etc.) but also some differences.

Note this course does not cover the Node.js HTTP[|S|/2] modules – we have a separate detailed course covering these along with Express and PUG, which is an ideal follow-on course to this one.

Contents of One-Day Training Course	
Target Audience Experienced web developers who wish to get up to speed developing for the Node.js runtime using TypeScript. **Prerequisites** No previous experience of Node.js required. General experience with web programming is required. All demos and lab exercises will be in TypeScript, so attendees need to know TypeScript.	**Feature Tour of Node.js 22** Node.js 22 is the latest edition of this very popular server-side runtime for web, CLI and other workloads – let's see what Node has to offer **Node And TypeScript** Most Node apps up to now have used JavaScript – we will use TypeScript Benefits of TypeScript for server coding Using Node from TypeScript (.d.ts files) **Launching Node** Command line options for Node itself Environment settings Keeping a node app running **Event Loop** Review of how the event loop works The important role events play in Node The Events module EventEmitter Preference for asynchronous Role of listeners **Non-Blocking IO** Accessing the file system Accessing networks **Network Programming** DNS UDP/Datagram TCP using the Net module IPC servers using the Net module **V8** Google's V8 engine provides the core execution environment for node Using the V8 module

Additional right-hand content column:

OS Module
Portable access to capabilities/services of the operating system on which Node runs

VM
The VM module can be used to compile and execute code in a language VM

Utilities
Utilities module
TTY
Console
Timers
StringDecoder

Crypto
Cipher
Decipher
Handling certificates
Hashing

Streams
Stream types – readable, writable, duplex and transform
The readline module
process.stdout

N-API
Building portable C/C++ modules that work with Node using the new N-API

Advanced Node
Process and child processes
The cluster module
Shared server ports and child processes

CLI Project
Building a command-line app showing how to use many parts of the Node framework together in a realistic app

Browser Runtime Programming Using TypeScript
File API, IndexedDB, GeoLocation, Beacon, HTTP/x, Events, CORS, Fetch, Formats and Device Info

The programming environment inside the modern web browser has significantly matured and now offers a rich and diverse range of capabilities, some at the UI level and some at the underlying runtime level. Many of the newer features are currently not being fully exploited by web developers, who focus exclusively on the UI. This course aims to change that by exploring in depth the non-UI aspects of browser programming, using the TypeScript language for all demos and lab exercises.

The runtime programming APIs in a modern browser now rival what modern OSes offer. A web app has many natural advantages over mobile apps (here's four: the power of the URI, works everywhere, cloud-friendly, immediate app updates). In areas such as sandboxed file access, networking, data formats and device info, a modern browser offers the web application developer a comprehensive selection of functionality that, when used correctly, can easily compete with what is available for native mobile apps.

IMPORTANT: This course does not cover parallel (web worker) & asynchronous programming – we offer a separate full course that covers these topics in detail.

	Contents of One-Day Training Course	
Target Audience Web developers wishing to fully leverage the runtime (non-UI) capabilities of modern web browsers. **Prerequisites** Good experience of web development, including modern HTML. Knowledge of the TypeScript programming language.	**Modern Web Platform** Tour of the modern web platform (what we already know / what is less familiar) Major features and which browsers implement them (caniuse.com and iwanttouse.com websites) Dynamically detecting in code available browser capabilities and optimizing app **File API** File Sandboxing Accessing and updating files File metadata and raw blob content File[Reader|Writer|system] API **Indexed DB** A B-tree like persistence mechanism with powerful indexing that can serve as basis for in-browser client database Comparison with other storage options **Text Handling And JSON** Handling various text encodings TextEncoder and TextDecoder JSON parsing and generation Use of JSON in the browser **GeoLocation** "This specification defines an API that provides scripted access to geographical location information associated with the hosting device." **Beacon** Sending data asynchronously to server (e.g after page close) navigator.sendBeacon Setting priority	**HTTP/2 & HTTP/3** Latest generation of HTTP protocol Binary, multiplexed, full duplex, priority Server push New app architectural possibilities **Server Sent Events** How it works EventSource API Event streams Event handlers **CORS** Cross-Origin Resource Sharing Role of origin in HTTP protocol cross-origin access CORS preflight request and CORS request **Fetch** A significantly improved replacement for XmlHttpRequest "The Fetch standard defines requests, responses, and the process that binds them: fetching." (WHATWG) **Data Formats** Brotli Compressed Data Format Data URIs using base64 encoding data-* attributes **Device Info** Accessing device details Vibration API Battery status **Project** Bringing together the ideas covered in this course to design and build a specialist runtime engine to execute in a browser

Asynchronous, Parallel & Reactive (RxJS) Programming Using TypeScript
Managing time, multiple workers and data streams

[Sample: internals] Coordinating multiple activities is one of the most difficult areas of advanced application development. In this course we explore all the different options open to developers and see how they can be integrated into modern TypeScript applications.

All modern hardware (even low-end mobile devices) support multiple CPU cores and allow parallel code execution. All have timers that allow asynchronous workloads to be queued for execution in the future. It is up to app developers to exploit the varying capabilities of hardware available to them to deliver optimum apps.

This course is ideal for TypeScript developers who wish to more tightly manage how their code and data are processed - where, when and in what order.

All samples and labs in this course use TypeScript as we think it is best for larger applications – much of what is covered is useful to JavaScript developers too.

We explore code both running on the server (e.g. as part of a Node.js server application) and in the modern browser (as standalone TypeScript code, or as part of a larger Angular application).

	Contents of One-Day Training Course	
Target Audience TypeScript developers who wish to have more control within their apps over time, multiple workers and observable data streams **Prerequisites** Understanding of asynchronous and multi-threaded programming from other environments, together with knowledge of TypeScript.	**Options for Async & Parallel** Review of all the possibilities available to TypeScript developers to manage time, distribute code across contexts to be executed and handle data streaming **JavaScript VM Event Loop** Need to fully understand the event loop and event ordering to optimize our code Adding events and consuming events **Promises & Timers** How a promise works: in real life / code Programming with a promise Error handling Using VM timers What setTimer(0, <func>) means Promises/A+ (https://promisesaplus.com) **TypeScript async/await** Making asynchronous source more readable while maintaining capabilities Use of async and await keywords Designing an asynchronous framework **Web Workers** A web worker is a thread Threads cooperate via message passing Creating and managing web workers Types of web workers Worker lifetime control [Dedicated|Shared]WorkerGlobalScope WindowOrWorkerGlobalScope **Messaging Between Workers** Message ports / Message parameters PostMessage / onMessage Organizing message flows	**Shared Array Buffer & Atomics** Shared memory within a browser Atomics provide synchronization to protect shared array buffers **Zone.js** Architecture of Zone.js Multiple zones can live within the same web worker or main browser context Creating and using zones Zone.js is heavily used in Angular - how? **RxJS Overview** Observerable / Observer / Subject Hot & cold next/error/complete Testing RxJS code using jasmine-marbles **RxJS Streams** An observable as a dual of an enumerable Subscriptions App architecture and stream processing **RxJS Operators** Processing individual elements via a large collections of operators What's new with pipeable operators Custom operators **Advanced RxJS** Notifications Schedulers (incl. new TestScheduler) Connectables **Project** Having explored the fundamental constructs for managing time, code execution and data streams, we conclude with a project that demonstrates all these ideas together

.NET 8 CLR Programming Using C#
Architecture, Assemblies, Type/instance, Attributes, Reflection, Security, Nuget, RegEx, Unmanaged

The .NET 8 Common Language Runtime (CLR) is the foundation for all aspects of the modern .NET initiative. It offers leading edge services for applications on Windows, macOS, Linux and more. The .NET provides a "managed" runtime environment in the sense that an in-process runtime engine, separate from your code, to assist with its execution. Its modern feature-set includes a system-wide object model, full control of memory management, granular security, innovative metadata, sophisticated type loading, and virtualized contracts between types (so that their physical layouts may be defined / optimized at runtime).

Regardless of which .NET application type you are building – rich client UI, web UI, web service, class library, Windows Service, console - you must understand how it works with the CLR and how it can programmatically interact with the capabilities of the CLR..NET 8 works with the CLR to provide a rich .NET Standard-based class library.

.NET 8 is becoming the platform of choice for new projects. It is highly regarded and is gaining a reputation as a solid foundation for innovative solutions.

Contents of One-Day Training Course	
Target Audience System architects and senior software engineers who need to rapidly get up to speed with .NET CLR programming. **Prerequisites** General understanding of high level .NET concepts. Attendance at our *C# 12 Language* course or equivalent experience is needed. This course covers .NET 8 using C# 12 and Visual Studio.	**Overview** Multiple implementations of .NET .NET Standard is a specification that mandates how all should work .NET Framework: stable, mature (20yrs) .NET Core (1/2/3): Win/macOS/Linux .NET 5/6: New merged approach Modern – e.g. WebAssembly integration For new codebases, choose .NET 8 **CLR Architecture** Feature-rich runtime for many languages What capabilities does it provide? **Assemblies** An assembly is a (DLL-like) delivery & management unit for CLR types Strong names / versioning Modules and assemblies Locating assemblies **Types** Importance of type / CLS & CTS Types and different languages System.Object & identity Nullable Annotations **Advanced Types** Methods – virtual, abstract, overloaded Type hierarchy / relationships / generics Properties, enumerations, constants, fields **Instances** Types & instances Reference type and value type Finalisation / IDisposable Garbage collection / stack and heap Calling methods/anonymous methods	**Compiler Options** Native compiler vs. JIT - contrast Using .NET Native .NET on macOS and Linux **Nuget** Using .NET's package manager Creating your own packages **.NET 8 API** Extensive quality class library Solves problems with .NET Framework Portable; small; new features (e.g. SIMD & AssemblyDependencyResolver) Nullable Annotations **Config** Externalized Configuration Handling configuration data Format of config files & custom settings Application and machine configuration **Custom Attributes** Advantages of using attributes in IL Pseudo-custom attributes Creating & detecting attributes **Metadata & Reflection** What data is stored along with code Manifest & metadata tables Peaking inside an assembly Discovering types **Security** Identity features of .NET security Certificates & permissions Command-line and UI security tools **The Unmanaged World** P/Invoke & how to call C libraries

.NET 8 Multithreaded & Parallel Programming
Concepts, Kernel Objects, Threads, Synchronization, Tasks, TPL, PLINQ, Parallel Collections, TPL Dataflow

[Sample: presentation] This course examines how to use .NET 8 to build sophisticated architectures using multithreaded and parallel programming. When designed correctly, these can substantially increase application performance and responsiveness to distributed clients and end-users. The kernel object is the opaque foundation for Windows multithreading – based on this are .NET constructs for the process, thread & various synchronization objects - mutex, event, semaphore, waitable timer and more – each of which targets different needs. Thread activity, lifetimes and inter-thread communication must be co-ordinated.

A server must efficiently multiplex many I/O requests over a few threads – which is precisely the goal of threadpools in .NET 8. We see major benefits from the addition of parallel programming to .NET, especially with TPL, PLINQ, concurrent collections and TPL Dataflow. We explore the critical role of `Task` in this new approach. We see scope for custom enhancements to TPL in a variety of scenarios.

This course supplies attendees with a clear understanding of .NET multithreaded and parallel programming, together with experience of their use.

	Contents of One-Day Training Course	
	Multithreading Concepts	**Parallel Programming in .NET**
	Thread definition	Takes a higher level view of concurrency
	Scheduling vs. synchronization	Parallel querying
	Parallelism and concurrency	Parallel algorithms and supporting types
	Compute-bound and I/O bound apps	**Task**
Target Audience	Race conditions, deadlock, starvation,	What is in System.Threading.Task
System architects and	priority inversion	Detailed look at Task class and how to use
experienced developers	**OS Foundation**	Parallel / TaskFactory / TaskExtensions
who need to gain an in-	**(e.g. kernel objects)**	**PLINQ - Parallel Linq**
depth understanding	Win32 kernel objects	Applying the ideas of Linq to Objects
of .NET 8 multithreaded	Usage counting / Kernel object handles	across multiple threads
and parallel programming.	Sharing handles among processes	Query operartors and threading
	Managed Threads	Additional capabilities of PLINQ
	OS threads and managed threads	**Concurrent Collections**
	Creating a new managed thread	Original .NET collections are not thread-
	The ThreadStart delegate	safe, and why this can be good/bad
	Thread priority and processor affinity	New concurrent collections offer similar
	Thread management & lifetime	API surface to original, but now threadsafe
Prerequisites	**Synchronization**	Deep dive: System.Collection.Concurrent
Attendees must have	Monitor (C# lock), Mutex, Event, Timer	**Advanced Tasks & PLINQ**
experience of systems-	Waiting (once, many), SpinWait	Schedulers
level programming.	The "Protect data, not code" principle	Lambda expressions
	The Interlocked class / Semaphore	Custom operators
Attendance at our *.NET 8*	**Memory, Threads and SIMD**	Partitioners
CLR Programming Using	Thread data slots	**TPL Dataflow**
C# 12 course or equivalent	VolatileRead/Write	Library of dataflow constructs
experience needed.	SIMD - Single instruction, multiple data	Message passing
	Intrinsics	Based around composition of blocks
	Managed Thread Pool	Very useful for certain kinds of workloads
	The thread pool is a runtime-managed	**Multithreaded Project**
	pools of threads for processing I/O,	A complete multithreaded embedded HTTP
	work-items and timer handlers	web server that uses a thread pool to
	Architecture of large multithreaded app	efficiently manage very many requests

.NET Managed Extensibility Framework (MEF)
Overview, Concepts, Export & Imports, Metadata, Catalogs, Providers, Contract Adapters, Project

The Managed Extensibility Framework – MEF – is a powerful composition framework from Microsoft for the creation of extensible applications composed from parts supplied at runtime. MEF is an open-source project and in its latest iteration is supplied via nuget and is documented as part of the .NET Platform Extensions.

MEF is suitable for use with large-scale modular apps (the Visual Studio editor itself combines its components using MEF). Application architects are being asked to provide designs exhibiting high levels of extensibility and flexibility and MEF is a key building block.

MEF allows you to create parts (i.e. components) using managed code that identifies imports (what the part needs from other parts) and exports (what the part offers to others). Parts come from catalogs and the composition process is managed by various export providers and composition services.

This course is your first step in getting up to speed with the MEF way of dynamically composing apps from components. Attendees will learn about the extensibility concepts in MEF and get practical guidance on how to use MEF when designing custom own apps.

Contents of One-Day Training Course	

Target Audience
Senior software engineers wishing to author composable parts using MEF.
 	
Prerequisites
Attendees are expected to be experienced C#/.NET developers having familiarity with modern object-oriented design concepts.
	Overview
	How best to tackle extensibility?
	Relationship to IoC
	What composition delivers
	Developer setup
	Concepts
	Composition
	Parts
	Imports
	Exports
	Contracts and metadata
	Adapters
	Export providers
	Imports & Exports
	Attributes
	Single and multiple exports
	Describing exports / ImportMany
	Eager and lazy export loading
	Metadata
	Dictionary of name/value pairs
	Strongly typed metadata
	CreationPolicy
	Catalogs
	Role of catalogs is to supply
	   parts and definitions
	TypeCatalog
	AssemblyCatalog
	DirectoryCatalog
	AggregateCatalog
	Composition
	CompositionContainer
	PartNotDiscoverable attribute
	Initial parts
		Export Providers
		Role: to provide exports
		CompositionContainer manages a topology
		   of export providers
		Flexibility in supplying parts
		Defaults/features in product line/multi SKU
		Contract Adapters
		Concept of contract adapters
		Why & how to adapt between contracts
		Contract name and metadata constants
		Writing an adapter
		Primitives
		Definitions – Part, Import/Export
		ContractBasedImportDefinition
		ComposablePart
		ComposablePartCatalog
		Error Handling
		ICompositionElement
		Custom exceptions
		Composition errors
		Debugger proxies
		Debugging Session
		Debugging app through the MEF source to
		   see where errors are raised and handled
		Designing MEF Applications
		Think about the MEF cascade
		Batch of top-level parts
		App-internal parts from AssemblyCatalog
		Extensions from DirectoryCatalog
		Project
		We examine a project that explores
		   extensibility models and implements
		   important concepts covered in this course

ML.NET
Overview, Architecture, Estimators, Training, Predictions, TensorFlow, Performance, Data, Project

It may come as a surprise to many but 80% of the codebase of a typical machine learning application has nothing to do with machine learning. A typical ML application needs code for multithreading, database access, lots of mathematics, network connections, configuration files, containers etc. and all this is common to any kind of application. This is important to keep in mind when selecting a ML framework – as it is one (of two) key reasons many C#/.NET developers pick Microsoft's ML.NET as their ML framework of choice. The other reason being that ML.NET is simply an excellently designed ML framework.

ML.NET is an open source project with a GitHub repo, that is undergoing rapidly enhancement, as shown by the very frequent commit cadence.

This course helps C#/.NET developers use their existing skillset and managed codebases to create modern ML applications. Machine learning is often seen as complex from the outside by developers not familiar with the area, but when approached in small incremental steps (as we do in this course), it becomes a gradual learning experience. The key as we see is to get to running code quickly and then gradually expand capabilities.

	Contents of One-Day Training Course	
Target Audience C#/.NET developers wishing to create machine learning applications for managed code. **Prerequisites** Knowledge of C# 12 and .NET 8. Though ML.NET provides excellent implementations of what we need, it is still useful to have a high-level understanding of what is going on. Hence some foundational knowledge of mathematics is required, as machine learning is so heavily mathematics-based. (MLNET does implement the mathematics we need).	**ML.NET Overview** "ML.NET enables machine learning tasks like classification ... regression ... and many other ML tasks such as anomaly detection, time-series-forecast, clustering, ranking, etc." [link] **ML.NET Architecture** High performance meets ML meets .NET Tour of major components DNN Transforms Learning algorithms **ML.NET Tooling** Installing ML.NET: `dotnet add package Microsoft.ML` ML.NET Model builder / ML.NET CLI Use with Visual Studio 2022 **Context and Data** MLContext is the connection between your app's code and the ML.NET engine Creating and using a context Context configuration ML involves lots of data – so need to examine how to supply data (bulk, streamed, other) to the ML engine **Estimators** Role of estimator-"untrained transformer" `IEstimator` and its implementations Idea of fit **Transforms** Image Onnx TimeSeries	**Training And Predictions** What is involved in training a model Using trained model Constructing prediction engine Making predictions **Calibrators** Using calibrators for binary classification ICalibrator Calibrator implementations (native, isotonic, platt) **Integration with TensorFlow** TensorFlow is an excellent ML framework One can call it from ML.NET-enabled apps What extra functionality does it provide? Other integrations **Testing ML Code** Often ignored, how to test ML code is crucially important item for production-quality delivery Performance issues **ML.NET App Design** Going beyond the APIs, exploring what is needed to design an ML enabled .NET app ML.NET role as part of larger app design Certain important design considerations **ML.NET Internals** ML.NET is written in C# and it is quite informative to read and understand the source tree **Project** Exploring steps need to add machine learning capabilities to an existing C#/.NET application using ML.NET

Java 22 Runtime Programming
IO, Processes, Serialization, Asynchronous, Security, Networking, Web Access

After developers learn the Java 22 language they next must learn about the Java 22 runtime environment and the APIs it provides. Java and its add-on packages offer a vast range of APIs and often it can be daunting for developers new to Java to figure out what goes where. Initially, to simply get work done for their specific assignments can be a challenge. This course aims to overcome this and takes developers already proficient in the Java 22 language on a walkthrough of common scenarios – we look at relevant APIs and the runtime ideas underlying them and help attendees write code efficiently and become productive as Java devs.

Beneath Java on every implementation is an OS, whose capabilities are exposed to Java applications via an API. The Java runtime itself, known as the JVM, adds additional capabilities. Base class libraries and layered libraries offer even more functionality. Taken together, a rich multi-layer of readily available functionality is provided for application developers to exploit in their own applications.

The aim of this rapid-paced course is to cover as much as possible of the fundamental APIs that devs need and provide them a good grounding in practical API usage.

Contents of One-Day Training Course		
Target Audience Developers wishing to create libraries and applications using Java 22's runtime capabilities. **Prerequisites** Attendees must already have attended our *Java 22 Language* course or have similar Java language programming experience. Note: This course does not cover Java multithreading. We offer a separate complete course on this topic.	**Overview of Java Runtime** Documentation and tooling How everything works together Overview of module layout Interaction with the JVM Tour of all major runtime features **Garbage Collector** Interacting with the GC Impact of various approaches Z Garbage Collector **java.io** File handling File and string readers and writers Buffering java.util.zip **Serializable** Serializing and deserializing an object NotSerializableException Object stream APIs **Managing Processes** Process class represents a process Creating a process with `ProcessBuilder` Launch mechanisms – e.g. `VFORK` and `POSIX_SPAWN` Redirect via `ProcessBuilder.Redirect` **Asynchronous I/O** Java async I/O design pattern Streams Asynchronous channel APIs **JMS – Java Message Service** Rich message exchange framework Reliable / asynchronous Point to point vs. pubsub	**Utilities** Text handling & regex Internationalization (e.g. new Unicode 11) Time / mathematics / etc. **Java And Security** A comprehensive security framework authentication, authorization, auditing Java and PKI Use of cryptographic algorithms **Advanced Security Features** SecurityManager Keystore Code security – code signing, bytecode verification, avoiding common threats **Java Networking** Socket programming with Java specifying network addresses Socket options Creating UDP and TCP connections **Web Access** HTTP 1.1, HTTP/2 (JEP110) and HTTP/3 URI SSL/TLS **Reflection** `java.lang.Object.getClass()` How to use `java.lang.Class` Reflection namespace – java.lang.reflect Constructor, Field, Method, Parameter **Additional Libraries** java.instrument and logging Transactions Java Management Extensions (JMX) Java Naming & Directory Interface (JNDI)

Java 22 Multithreading
Concepts, Kernel, Threads, Synchronization, Concurrent Collections, Debug, Akka, Server Design

[Sample: snippets] This course examines how to use Java 22 to build sophisticated multithreaded architectures. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. The kernel provides a number of opaque objects that are the foundation for multithreading – based on this are constructs for the process, thread & various synchronization objects – reentrant locks, event, semaphore, waitable timer and more – each of which targets different needs. Thread activity, lifetimes, inter-thread comms and memory usage must be co-ordinated.

Various higher-level design patterns may be used to route workitems in multithreaded servers. Tools may be developed to determine which thread is blocked waiting on which resource, and the state/owner of each resource. A delegate-based configurable pipeline + a cache are often appropriate. The optimal server architecture is one active thread per processor core. A server must efficiently multiplex many I/O requests over a few threads – which is precisely the goal of threadpools in Java 22. This course supplies attendees with a clear understanding of the concepts underlying multi-threading, together with experience of their use in Java.

Contents of One-Day Training Course

Target Audience
System architects and experienced developers who need to gain an in-depth understanding of Java multithreading.

Prerequisites
Attendees must have some experience of systems-level programming.

Attendance at our *Java 22 Runtime Programming* course or equivalent experience needed.

Multithreading Concepts
Scheduling vs. synchronization
Parallelism and concurrency
Compute-bound and I/O bound apps
Race conditions, deadlock, starvation, priority inversion

OS Foundations
Processes & threading in various kernel
Usage counting
Sharing handles among processes

Threads in Java
Java's threading architecture & lifecycle
The Runnable interface
ThreadLocal – each thread has its own
 copy of the variable / ThreadGroup

Creating Java Threads
OS threads and Java threads
Creating a new thread: java.lang.Thread
Implementing a Runnable
Extending the Thread class
Thread priority and processor affinity

Synchronization
Locks, mutexes (reentrant locks),
 semaphore, events and timers
Waiting (once, many), idea of spin wait
The "Protect data, not code" principle
Atomics with java.util.concurrent.atomic
A synchronized block

Java Collections And Threads
When do we need to protect collections?
Selecting and using thread-safe collections
java.util.concurrent.locks
java.util.concurrent for concurrency

java.util.concurrent.atomic:*
 supports lock-free thread-safe
programming on single variables

Thread Pool
The thread pool is a runtime-managed
 pools of threads for processing I/O,
work-items and timer handlers

Debugging with Threads
Querying information about running
 processes/threads and their attributes
Thread tracing/debugging in tooling

Resource Management
Creating a custom resource browser, to
 display which thread is waiting on which
synchronization resource

Design Issues
Single writer/multiple readers, once-off
 initialization, Dining Philosopher
Converting legacy code to multithreading

Multithreaded Architectures
Pipeline, Producer-Consumer, Work-Crew
 and Master-Slave Models
Create threads on demand vs. elastic pool

Akka – The Actor Model
Moving beyond foundational thread
constructs, explore higher level frameworks
Akka is based on the actor model and is
 highly suited to large scale projects
Working without locks – how?

Multithreaded Project
A complete multithreaded embedded HTTP
web server that uses a thread pool to
efficiently manage very many requests

Technology / Data School

Query
- RDF-OWL-SPARQL
- SQL
- XML
- .NET LINQ, Expression Trees And Rx
- STL using C++

ORM
- .NET EF Core 8 Using C#

Format
- PDF Programming

Repository
- GIT and GitHub

Fundamentals Of Storage
Disk Hardware, Connectors, OS Storage Subsystems, File Systems, RAID, Tx, Caching, Hashing, B-Trees

This course provides a comprehensive tour of the important storage fundamentals that all technologists need to understand in order to build, provision and operate modern IT solutions that involve storage (which of course means all such solutions). There's a lot more to storage than the hard disk on a PC or simple APIs such as fopen() / fwrite() that developers might use. In this course we explore storage end-to-end and see how it interacts with remote disks, hypervisors, the cloud and various in-memory representations. We see how the distinction between storage, database and memory is being blurred. We see how system-wide reliability and performance demands impart on storage. We see how modern OSes and their powerful storage stacks allow rich journaling file systems and databases to be built that deliver a wide variety of enhanced storage features. We also explore specific technical approaches to transactioning, caching, hashing and B-Trees.

A good understanding of storage fundamentals along with clear knowledge of the storage-related technical options available to choose from helps all involved in engineering to make the optimum design decisions.

	Contents of One-Day Training Course	
	Storage Concepts	**Modern File Systems**
	Latency	Journaling
	Throughput	128-bit sizes
	Reliability	Virtual File Systems And Client File APIs
	"bit-rot" and other issues	Detailed look at internals of a modern file system – ZFS
Target Audience	**Modern Storage for …**	
Developers, devops, IT professionals, engineering managers – all of whom need an understanding of the core building blocks of storage platforms.	File	**RAID**
	Object	Costs and benefits of various RAID types
	XML	Stripping
	SQL	When errors occur in hardware and in software
	Graph	
	SCSI – Talking To Storage	**Transactioning**
	Small Computer System Interface (SCSI) is a comprehensive protocol / command / interface between controller and disk	Transaction principles
		Two phase commit
		Hierarchical transaction groups
Prerequisites	**Disk hardware and connectors**	Savepoints
Good all-round knowledge of modern computing infrastructure at a technical level.	Traditional disk	Resource manager
	SATA / SAS	Enlistment
	SSD and Flash	**Caching**
	NVMe and PCI Express	Putting data close to where it is used
	RAM disk	Cache lifetimes
	Tape, Optical	Keeping data in cache fresh
	Managing larger amounts of data	Caching architecture
	Tape and optical formats used for backup, transporting and long-term archiving	**Identity And Hashing**
		How to identify storage objects
	DAS, NAS, SAN	Role of hashing algorithms
	Directly Attached Storage	Importance of hashing for de-duplication and other aspects of storage
	Network Attached Storage	
	Storage Area Network	**B-Tree**
	Storage Subsystem In An OS	B-Tree is the most important data structure for storage architectures
	Storage and networking are the largest subsystems inside any OS	Performance characteristics of B-Trees
	Exploring an OS storage architecture	Variations of B-Tree layout

STL: The C++ Standard Template Library
Algorithms, Containers, Iterators, Functors, Adaptors, Allocators, Performance, Internals

STL is an amazing masterpiece of software engineering. In addition to learning about a rich object collection system, developers will profit from studying STL deeply as they will learn how to put together their own modern frameworks to comprehensively tackle their specific needs.

The goal of this course is to bring C++ developers up to speed with all aspects of STL programming. We start with a tour of what STL has to offer and how it builds on some of the latest ideas in modern C++. We explore all the technical constructs defined by STL.

Iterators are used to flexibly define sequences of elements and manage the navigation among elements, without exposing the internal arrangement of elements. Algorithms are the operations we wish to perform on sequence of elements (independently of how those elements are stored). Containers (both sequence and associative) are how storage of groups of elements is managed. Adaptors provide more specialist access to containers. Allocators are used for memory layout. We also examine the very interesting STL architecture and discover how some of its design ideas used internally may be applied in designing our own class libraries.

	Contents of One-Day Training Course
Target Audience C++ developers who wish to learn about the power of STL and see how to use it aggressively in their own applications **Prerequisites** Good knowledge of the fundamentals of C++ programming, especially templates and memory management.	**STL Overview** Tour of STL capabilities Generic programming Visiting with iterators Generic containers Container-independent algorithms Application programming using STL Highly efficient and flexible solutions **C++ Review** Review of aspects of modern C++ (e.g. templates, memory) that STL leverages **Iterators** A specialist pointer to an element Category Element type & distance type Reverse/stream/insertion iterators **Algorithms** Operates on sequences Passing in sequences using begin-/end- Template-based functions to perform ops Review of available algorithms Functors as algorithm predicates **Sequence Containers** Containers are collections Ordered collection of elements deque, list, vector Random access vs. sequential access **Associative Containers** Elements and their associated keys map, set, bitset **Adaptors** Adapting a container to a specialist need e.g. look at stack/[priority|]queue interface **Allocators** Use default allocators at first, expand later Various strategies for managing blocks of memory using <memory> header **STL & Shared Libraries** Issues when passing STL collections across DLL/.so shared library boundaries Need for using same binary layout **Deploying STL** Optimizing STL use in your own projects Container selection – different containers have differing performance capabilities and differing feature sets Algorithm selection – being aware of large range of algorithms available is important **Custom** Building custom: - containers - algorithms - iterators - adaptors - allocators **Design Ideas** Review of architecture of STL Incorporating ideas from STL into your own framework designs **Internals** STL is delivered as a set of header files Exploring how it is put together More specialist functionality **Project** Building a C++ project that uses STL extensively

.NET LINQ, Expression Trees And Rx
Lambda/Query Expressions, LINQ-To-Objects, Trees, Providers, Rx Observ[able|er], Subjects, Operators

.NET's LINQ (Language INtegrated Query) is an innovative querying capability built into the .NET Framework and languages such as C#. Querying plays a major role in most applications. For .NET and C# to pay particular attention to how code can query a variety of data sources (e.g. SQL databases with EF Core, XML documents, observable streams with Rx, collection classes, management objects, ..) can be most beneficial.

Once .NET application developers learn the basics of LINQ they can then explore how to extend it and how to expose custom data sources as LINQ targets.

Think of .NET expression trees as an AST of a query. They are used in LINQ and many custom solutions. When LINQ providers talk to remote data stores, expression trees are at the heart. Expression trees can automatically be created by a .NET compiler and also directly interacted with by application code.

Rx (Reactive extensions) is "an API for asynchronous programming with observable streams" [link]. It is based on the observable pattern (one of the GoF patterns). Rx is available for many languages (e.g. Angular uses it extensively), this course covers Rx+C#.

	Contents of One-Day Training Course	

Target Audience
Software engineers wishing to learn about the foundation of LINQ, Expression Trees and Rx.
 	
Prerequisites
Attendees are expected to be experienced C#/.NET developers with some awareness of data querying, collections and abstract syntax tree usage.
	Overview: LINQ, Expression Trees and Rx
	Introduction to each, how they fit together and how they might be used in applications
	Functional programming ideas
	Review of C# Features
	Modern features of C# that are of interest
	Lambda expressions, anonymous types, extension methods, var, flexible object initialization, partial classes/methods
	Func<> and Action<>
	LINQ Concepts
	Immediate & deferred execution
	Query Syntax vs. Method Syntax
	What are operators
	Use Query Expressions
	Query Expression Syntax
	SQL-like format
	Typing
	Standard Query Operators
	Tour System.Linq namespace
	Exploring provided operators
	Adding custom query operators
	Inputs and outputs
	Moving to saying what you want
	Lambda, delegates and expression trees
	IEnumerable<T> vs. IQueryable<T>
	Designing LINQ Providers
	How is the network involved?
	How does data flow?
	Where, and when, are expressions run?
	Creating a custom LINQ Data Provider
	Programmatically querying custom data
		Expression Trees
		Introduction to expression trees
		Expression trees and IQueryable<>
		Dynamically compiling expression trees
		Review of important expression types
		System.Linq.Expressions
		The LambdaExpression type
		Factory methods
		Debugging
		How to debug LINQ code
		Writing your own debug visualizer for expression trees
		ReactiveX for C#
		Observable and Observer
		Enumerable vs. observable
		Rx operators
		Subject
		Disposables
		How to use Rx within application code
		Advanced Rx
		Rx and ..
		  Handling time / Exceptions
		  Threading / Subscriptions
		  Notification / Materialization
		  Cold vs. hot / Scheduling
		Async Streams
		Important improvements to async/await allows precise control of async streams with rich new features
		Custom Rx
		Building your own observers, observables and operators
		Managing subscription lifetimes

RDF, OWL And SPARQL
Ontologies, Standards, Triples, Datatypes, RDF, RDF Schema, OWL, Entities, SPARQL, Reasoning

This course provides a guided tour of the world of knowledge representation and reasoning, from an ontology perspective. We will see that a graph is the most scalable, extensible and distributed form of knowledge representation and facilitates knowledge reasoning to incorporate additional facts.

W3C has defined a layered set of standards related to defining and querying ontologies. We will discover how each standard builds on a foundation to add more ontology-related functionality and together they define a comprehensive solution to ontology management.

Resources are identified with internationalized URIs (IRIs). We start with an agreed set of basic datatypes (number types, strings, URI, dates), then use them to define simple statements in the form of triples (subject-predicate-object), then build groups of such statements into graphs, and then combine multiple graphs into datasets.

We add a query engine in front of such datasets to make them accessible to distributed clients. We can define rules to add more control over ontology interaction. We support knowledge reasoning via inferencing.

	Contents of One-Day Training Course	
Target Audience Software engineers and knowledge engineers who wish to learn more about creating and interrogating ontologies using the latest W3C standards **Prerequisites** A software engineering background with some experience of creating semantic models	**Ontologies** Representing knowledge The knowledge graph Description Logics (DL) Reasoning about knowledge Tour of the world of ontologies **Ontology Standards** W3C has been very active in defining a suite of standards related to ontologies Review of W3C layered standards **Protégé Tooling** Stanford University has created the Protégé tool (https://protege.stanford.edu) - "A free, open-source ontology editor & framework for building intelligent systems" **XML Datatypes** The common primitives (int, string, date) for ontologies and the knowledge graph Value space vs. lexical space Facets for specialization **Introduction to RDF** Role of Resource Description Framework Statement: subject, predicate, object Managing triples **RDF Schema** Extending the RDF vocabulary Defining classes and their properties Reification **Introduction to OWL** Web Ontology Language (OWL) expands the vocabulary for representing knowledge Tour of OWL capabilities – axioms, etc. Literals/datatypes/dataranges/expressions	**OWL Entities** Named Individual Class Datatype Object Property Data Property Annotation Property **SPARQL Introduction** SPARQL is to ontologies what SQL is to a relational database – a flexible language to query and update knowledge graphs **Advanced SPARQL** Result formats Update Federated queries **Existing Datasets** Exploring available big datasets, e.g.: - Dbpedia (http://wiki.dbpedia.org/) - Wikidata (https://www.wikidata.org) **Reasoning & Rules** Inferencing over ontologies Discovering new relationships Defining rules using RIF **Ontologies & Machine Learning** Some folks think of ontologies as competing with ML – we look at this question and how to use them together **Ontology Design** Review of how to create ontologies using what we have learnt **Project** Designing a large software solution incorporating ideas explored in this course

Structured Query Language (SQL)
Relations, SQL Overview, The Table, DML, Joins, DDL, Stored Procedures, Transactions, ORM

Structured Query Language (SQL) is the very popular purpose-built query language for relational databases. A mathematical relation forms the logical underpinning for a relational table. A table consists of an unordered set of rows; each row consists of an unordered set of columns. Columns are named and are associated with a datatype. Some columns has specific characteristics (e.g. primary/foreign keys). Indices can be attached to some constructs which greatly improve performance.

Whether building cloud, enterprise or mobile solutions, devs should at least consider and often select SQL as the basis for their data management needs. It offers a wide gamut of capabilities and is massively supported across all computing platforms.

There are many SQL products in the marketplace. This course explores standard SQL. Each database vendor adds its own nuances to this standard language, but all support a large core set of SQL functionality, and it is this core that we cover in this course. After this foundational course developers will be ready to explore the product-specific documentation for their selected SQL engine & selected app programming language.

Contents of One-Day Training Course

Target Audience Developers who wish to learn SQL by focusing on the large set of SQL features that is common to modern relational databases.	**SQL Overview** The mathematical relation Representing knowledge via tables Query language for relations Parameterized queries Stored procedures **Role of SQL** Tour of SQL in modern database engines, both client/server & library-based, such as: - Microsoft SQL Server - Sqlite - MySQL - Oracle **Defining a Table** Datatypes Primary key Foreign key Normalization **Intro to DML** SELECT INSERT UPDATE DELETE **Advanced DML** WHERE ORDER BY DISTINCT Aggregates (AVG, SUM, MIN, MAX) **Joins** Combining statements in a join Inner join Left/right join Full join	**DDL** CREATE and ALTER DROP TRUNCATE **Stored Procedures** Creating and calling stored procs Parameters and local variables Additional programming constructs (BEGIN/END, TRY-CATCH, etc.) **ACID** Atomicity Consistency Isolation Durability **Transactions** Unit of work All or none Begin Tx and commit or rollback **SQL And Security** SQL injection attack Dangers of dynamically building queries GRANT and REVOKE **ORM–Object Relational Mapper** Many object-oriented programming languages use an ORM to manage SQL queries - how do they work? **SQL vs. NoSQL** Contrast SQL and NoSQL approaches to data storage and data management Suitability for different scenarios **Project** Use of SQL as the basis for data in a comprehensive enterprise solution

Prerequisites
Experienced software engineers with some data manipulation background.

No previous SQL experience is required.

eXtensible Markup Language (XML)
XML InfoSet, XML Serialization, Namespaces, DOM, XML Schemas (XSD), XPath, XSLT, XML in App Design

XML is well supported on all technology platforms by many editors, tools and framework vendors. As it has proven its worth in the field, it has become a highly desirable feature to leverage in applications. Hence it is now a mandatory part of the skills set for modern software developers. The W3C has defined a cohesive series of XML standards, covering core information modeling, how data is to be serialized, a Document Object Model defining a programmatic API, how data is transformed and many more standards. XML has becoming the foundation for whole swathes of functionality in various computing environments.

Document file formats (such as ODF and Office Open XML) have standardized on XML. Many vertical industries (e.g. http://startndc.iata.org/ or http://hropenstandards.org) have defined custom XML schemas. Data exchange is exploiting it. Data delivery over the Internet is using it.

There are three reasons you will benefit from attending this training course. Firstly, you will learn what the XML data format is and its associated standards. Secondly, you will see how to integrate it with your own code. Thirdly, you will explore how XML can help you with app design.

	Contents of One-Day Training Course	
	XML Overview	**W3C XML DOM**
	XML is a metalanguage for describing other data languages	A Document Object Model for XML
	Representing data with markup	Programmatic Navigation
	Developer resources	Hierarchy of nodes
Target Audience	**The W3C XML Standards**	Fundamental & extended DOM interfaces
This training course targets component and web developers who need to know what XML is, how to program it and design support for it into their applications and web services	What are in the standards	Alternative: treating XML content as a stream - push (SAX) and pull (.NET)
	Layering	**XPath**
	Introduction to each standard	Identifying sub-sections of the XML tree
	The XML Information Set	XSL pattern matching
	Abstract description	**XSLT**
	Information items	Extensible Style Language Transforms
	Strict rules of XML	Transformations for display and into other data descriptions
	XML Serialization Standards	**Additional Standards**
	The fundamental XML structure is a tree	XML Base
	Each node in tree has a name, attributes & can be a parent of other defined nodes	XPointer / XLink
	Namespaces	XML Query
	Avoiding tag ambiguity when using multiple XML schemas	XML & Security
Prerequisites	Unique identifiers	**XML-Based Markup Languages**
Understanding of document and data storage needs, along with experience of Internet programming	Namespace aliases	Every industry needs to describe different data and hence need a different schema
	XML Structure	It is not feasible to have a single complete data description
	Defining what is permissible in XML	**XML in Application Design**
	Logical structure of information	How to design for XML
	Valid & well-formed XML	Creating a XML-based data format and programmatically loading and saving it
	XML Schemas	**Project**
	Describing metadata using XML	Case study showing the creation of a complex XML schema, it use within an app for data storage and exchange, and web delivery of data to a browser
	Defining schemas	
	Type system – simple and complex	
	Lexical space, value space and facets	
	Modularization	
	Schema Usage	

.NET Entity Framework Core 8 Using C#
Models, Annotations, Fluent API, Querying, Saving, Advanced, EF Core+Domain Model, Testing, Project

Handling data is one of the most complex aspects of any enterprise application development project. (The fact that Microsoft's .NET has frequently significantly changed it data story proves this!). Previous approaches have not been entirely satisfactory and many devs thought "There's got to be a better way". Enter EF Core. EF Core 8 is a modern ORM that has rapidly evolved out of code-first Entity Framework 7 for .NET Framework. The latest EF Core 8 (part of .NET 8) provides C# 12 app developers with a rich data access capability, in tune with modern development ideas (domain models, testing, SQL&NoSQL, agile, ..).

EF Core uses LINQ as its querying technology to great effect. LINQ is a radically different approach to querying that integrates object development and data access in a cohesive language environment.

This course covers the latest release of EF Core. The goal is to take C# / .NET application developers along a journey which will end up where they can competently program against a database (SQL or NoSQL) or other kinds of data sources from within their .NET applications.

Contents of One-Day Training Course	
Target Audience C# 12 / .NET 8 application developers wishing to create modern apps that need to access databases using the best ORM available for .NET, that is EF Core 8. **Prerequisites** Practical experience of C#, some previous database programming and SQL knowledge (any database). All demos and labs will be using .NET 8, C# 12 and Visual Studio.	**EF Core Overview** Overview of modern data access A powerful ORM (Object-Relational Mapping) for .NET applications Efficient, configurable, improving rapidly How EF Core fits in with rest of .NET **Tour Of Features** Use an initial sample app to practically demonstrate main feature set of EF Core Role of DbContext, DbSet, DbQuery, .. Data flows, config Error handling **Tooling** Development / data environment Connection string Distinct SQL RBMS data providers Non-SQL data providers Specialist data providers Additional developer tooling **Models** POCO – Plain Old C# Objects From these, auto-gen database schema Conventions / Annotations / Fluent API `OnModelCreating` Ways to influence generated model **Querying** Applying usual querying syntax: filtering, relationship following, ordering, aggregate Query types **Saving** SaveChanges / tracking / concurrency Transactions, cascades, keys Disconnected entities	**LINQ And EF Core 8** Deeper look at query language Going beyond basics, how to really use LINQ with EF Core Complex queries (e.g. joins) **EF Core and ..** Migrations Security Database views Stored procedures GIS **EF Core and Domain Model** Place domain model and data model in separate assemblies Data model references domain model (not the other way around) Domain model does not reference EFCore Prefer use of Fluent API – why? DDD Bounded Context = DbContext Repositories as interface in domain model (no EF Core) and implementation in data model (can be switched out for alternative) **Testing** This arrangement greatly helps testing Importance of dependency injection Modern testing approach and EF Core Testing database queries **Project** Developing an end-to-end layered solution consisting of Angular UI, ASP.NET REST API, domain model and EF Core 8 data model Looking particularly at last of these

PDF Programming
PDF for developers, Graphics, Imaging, Text, Fonts, Forms, Metadata, Security, Navigation, Project

Portable Document Format (PDF) is a standard representation of pages in a document. It is used for document interchange between document producers (word processors, report generators, CAD programs, specialist editing apps) and document consumers (on-screen viewers, printers, digital signature validators, forms engines, etc).

PDF became popular as the native file format for Adobe Acrobat and has been standardized by ISO, initially as ISO 32000-1:2008 (same as Adobe PDF 1.7) & recently ISO published a major update - ISO/PRF 32000-2.

Pretty much every computing device that has an attached display and/or printer can display/print PDF files. The vast majority of modern document editing environments can output to PDF - hence is is an extremely poplar shared representation of a publication. This course is aimed at developers who wish to gain a low-level technical understanding of PDF and how they can use it programmatically from within their own applications. This course provides an excellent foundation for developers who wish to build their own PDF import/export libraries and those who wish to use one of the existing libraries available for this purpose.

	Contents of One-Day Training Course
Target Audience Developers wishing to produce, transform and consume PDF files from their own applications. **Prerequisites** Software developers with experience of creating document editing apps, particularly their file formats. Knowledge of graphics programming (e.g. painter's algorithm) is required.	**PDF Technical Overview** Describing page representations **Color and Transparency** Applying ink to a page — Precisely specific color requirements / CIE / Masking / Clipping **PDF Concepts** — **Forms** Coordinate system — Forms options in PDF Painter's algorithm — Designing forms Imaging model — Accessing form fields after completion Color management — Changes in forms for ISO/PRF 32000-2 Transparency — **Metadata** **Document and Page Layout** — Attaching additional metadata to PDF Overall file structure — Metadata formats How each page is represented — Extensibility Contents of a page — **Digital Signatures** Common data — Signing what uses can see **Text And Fonts** — Digitally signing a PDF file Typography in PDF — Verifying a digital signature Glyphs — Cryptographic options (algorithms etc.) Fonts — **Navigation** Ways to represent text — Allowing users to easily navigate to pre-determined destinations within a document (hierarchy & thumbnails) Text objects — Document outline / Flexible inter-page navigation **Graphics** — **Multimedia** PDF supports many 2D diagramming constructs, from line to Bézier curves to various graphical shapes — Embedding video, audio and 3D in a document State information — Formats supported Use of 3D — Enabling media playing Path construction — **Project** **Images** — Write a project to programmatically create a PDF document Bitmaps and their use in PDF Supported imaging formats Rendering images on the page

Git and GitHub

Working Dir/Staging, Local/Remote, Clone, Push, Pull, Branch/Merge, Monorepo, GitHub Desktop

Source code is by far the most important asset any software company owns. It is more valuable than buildings, brand names, computer hardware, furniture or anything else a software company has. Source code needs to be valued and treated like the very important company asset that it is. Hence the need for a robust source code management system.

Git is the most popular source code management system; GitHub.com is the most popular Git cloud hosting solution. Either Git alone or Git and GitHub can be used to comprehensively manage and protect source.

Even if not using GitHub for their own source, app developers still need to get familiar with it as most of today's popular open source projects are using it and app developers will invariably need to use these.

This course covers both and helps developers gain hands-on experience in how to incorporate both into their development workflow. Many Git-related terms have entered the developer lexicon – push, pull request, cloning, forking, promoting, repo – and this course helps attendees understand each concept and mentally tie everything together to see how they work in unison.

Contents of One-Day Training Course

Target Audience
Software engineers and architects wishing to correctly manage valuable source trees

Prerequisites
Programming knowledge and some previous hands-on experience of any source control system.

For the GitHub part of the course, each attendee will need their own (free) GitHub login to complete the lab work.

Distributed Version Control
Using what you might already know
Adding distributed influence
Organizing teams via Git
Strategies for managing source trees
Terminology - push / pull, clone, fork, fetch, branching and merging
Lifecycle of a single line of code

Getting Started With Git
Installing and configuration
Simple usage
We explore where source can be stored (local and remote)
Promoting from working dir to staging and beyond
What happens during a commit

Working Locally
Init vs. clone
File system layout
The .gitignore file
Creating, modifying and deleting locally
Cancel changes (revert)
Logging/history/status

Branching / Merging
Creating and listing branches
Merging a branch
Change tracking / diff / Rebase
Feature branches vs. trunk development

Remote
Remote protocols
Connecting to remote repo servers
Push and pull commands
Fetch command

Command Line Tooling
Porcelain vs. plumbing
Beyond the basics - more complete look at advanced command line tools for Git

Managing as part of Toolchain
Git as part of toolchain
Use with other tools
Call via scripting (automated test runs, linting, Continuous Integration/Continuous Delivery - CI/CD)
What to do with generated info

Monorepo
Each project need not exist in separate repo
Multiple projects can be placed in a single repository – known as a monorepo
Practical ideas for using monorepos

GitHub
Source repositories in the cloud
Public (free hosting) & private (fee-paying)
Organizations and teams
Interacting with open source projects (issues, releases, changelog, pull requests)

GitHub Desktop
Enhanced (Electron-based) GUI to comprehensively manage Git repositories
Easy to set up and use

Git Internals
The source code for Git itself is an interesting read: https://github.com/git/git
What can we learn from exploring it?

Project
Organizing a large source tree using Git
Deciding on project and repo layout

Technology / Networking School

Protocol
- Fundamentals of TCP/IP Networking
- QUIC And HTTP/3 Protocols

Connection
- Protobuf and gRPC
- REST APIs – Designing, Specifying (OpenAPI)
  and Generating (OpenAPI Generator)
- ASP.NET Core 8 REST API Development

Orchestrator
- Kubenetes

Fundamentals Of TCP/IP Networking
Layering, Networking Architecture, Addressing, IPv4&IPv6, TCP, UDP, DNS, DHCP, HTTP, Security

TCP/IP is the dominant networking protocol suite and all involved in delivering modern technology solutions need a clear understanding of how it fundamentally works. Developers, DevOps and system admins, along with many specialists (such as those involved in database and security) could all benefit from attending this course as it will clarify the fundamentals of networking with the TCP/IP suite.

Networking touches on development, security, network traffic, identity, messaging, data transfer, resilience, middleware and lots more.

We examine the entire set of protocols in the TCP/IP suite. Even though the suite name mentions just two protocols, it is actually a much larger collection of protocols, with important protocols above and below the TCP and IPv4/IPv6 protocols.

In any modern OS, its networking stack is a large segment of the codebase and exposed API. By investigating what is actually happening between one app sending & another app receiving a message (by examining net traffic using appropriate monitoring tools), we can expand our understanding significantly.

	Contents of One-Day Training Course	
	Layers, Protocols and Devices	**UDP**
	Overview of networking participants	Comparison with TCP
	Networking big picture	"Best effort" datagram service
	Organization of the Internet	Protocol exchanges
Target Audience	Protocol development process (RFCs)	**DNS and DHCP**
Everyone with a technical background who is interested in how computer networks actually work.	**Use of Wireshark**	DNS architecture
	Eavesdropping on the network	DNS headers
	Extremely useful tool for low-level monitoring of protocol traffic	Record types (e.g. SRV records)
		DHCP architecture
	TCP/IP Family Of Protocols	**Internet Building Blocks**
	Layering of protocols	Routing protocols
	Distribution of responsibilities	Border Gateway Protocol (BGP)
	Packet structure (encapsulation etc.)	QoS
Prerequisites	**Data Link and Network Links**	AS
Experience of working on software projects, including development, deployment and ongoing service provision.	How octets are physically communicated technologies used at data and network	**HTTP 1.x & 2**
	Ethernet	HTTP basic messaging patterns
		Using Fiddler to examine HTTP traffic
	Addressing	Message chunking
	Approaches to network addressing	Multiple concurrent channels in HTTP/2
	Subnetting	**SMTP, POP and MIME**
No previous network programming or infrastructure experience is required, though any such knowledge would be beneficial.	ARP	Email protocols family
	Addressing for IPv6	Message stores
	IPv4 & IPv6	Representing emails
	Protocol exchanges	**Security And Networking**
	Packet structures	Firewalls
	MTU	NATs
	Internet Control Message Protocol (ping)	IPSec
	TCP	SSL/TLS
	Enhanced services	**OS Networking Stacks**
	Headers	Windows
	Windowing	Linux
	SYN packet and triple-handshake	Exposed C APIs (sockets etc.)
	Protocol exchanges	Networking APIs inside managed runtimes

QUIC And HTTP/3 Protocols
Transport/App Protocol, Connection, Packet, Frame, TLS 1.3, Multiplexing, Flow-Control, Implementation

QUIC and HTTP/3 are respectively the much anticipated next-generation transport protocol and next generation application protocol. HTTP/3 runs on top of QUIC. Both are currently undergoing standardization through the IETF.

Both protocols are being designed in unison to work extremely well together to achieve an elevated level of performance and security. They offer a number of very interesting new features and continue the work we first see in HTTP/2 of adding multiplexing for HTTP connections.

"The QUIC transport protocol incorporates stream multiplexing and per- stream flow control, similar to that provided by the HTTP/2 framing layer. By providing reliability at the stream level and congestion control across the entire connection, it has the capability to improve the performance of HTTP compared to a TCP mapping. QUIC also incorporates TLS 1.3 at the transport layer, offering comparable security to running TLS over TCP, with the improved connection setup latency of TCP Fast Open [link].
At the end of this course, attendees will understand how both protocols work and why they are important.

	Contents of One-Day Training Course	
Target Audience Networking professionals and senior software engineers who require a deeper understanding of these new protocols that will play a very significant role in the future of the web and the Internet **Prerequisites** Programming experience in any low-level language. Attendees will develop server and client-side implementations of both protocols as part of the labs. Good all-round networking knowledge; attendance at our *Fundamentals Of TCP/IP Networking* course or similar experience.	**Next Gen Protocols** What are we trying to achieve? Can we not reach these goals with TCP and HTTP/1.1 or HTTP/2? What do QUIC and HTTP/3 offer? **QUIC Overview** General tour of how QUIC works Based on UDP (a foundation that is already supported everywhere) QUIC plays role of TCP in protocol stack Message flows Connection set up and tear down Intro to security Available QUIC implemenations Extensions **Connections** Reasons for low-latency connection setup What impact this has? Connection migration Error correction Packet layout **QUIC Streams & Multiplexing** Unidirectional and bidirectional streams How multiple streams are multiplexed onto a single connection **Flow Control** Connection flow control Stream flow control **QUIC And Security** Modern TLS 1.3 built into QUIC Security is not an add-on option Review of security architecture Responding to NAT rebinding	**Frames** PADDING, RST_STREAM, [CONN|APP] _CLOSE, MAX_DATA, MAX_STREAM_DATA, MAX_STREAM_ID, PING, BLOCKED, STREAM_BLOCKED, STREAM_ID_BLOCKED, NEW_CONN_ID, RETIRE_CONNECTION_ID, STOP_SENDING, ACK, PATH_[CHALLENGE| RESPONSE], NEW_TOKEN, STREAM, CRYPTO **HTTP/3 Overview** Overall architecture HTTP/3 endpoints Options for discovery Types of streams (control, push, reserved) Available HTTP/3 implementations **HTTP Framing** DATA, HEADER, PRIORITY Settings Framing architecture **HTTP Message Exchanges** Message has one HEADERS frame and a number of DATA frames and optionally a concluding HEADERS frame Message flows **Connection Management** Cancellation, Compression, Prioritization Server Push Error management **Impact On Application Design** How these new protocols will influence application design (esp. multiplexing) **Project** Create simple implementations of QUIC and HTTP/3 and see how application code could benefit

REST APIs – Designing, Specifying (OpenAPI) and Generating (OpenAPI-Generator)
Design API, Write Spec, Autogen client/server code

A REST API is the most practical way of connecting clients and servers in a distributed heterogeneous world.

We start by discovering what is involved in creating high-quality REST APIs and how to best go about designing them, in a contract-first manner. Later in the course we also explore advanced API design topics.

Then we need to specify the API. OpenAPI is essentially a well-written standard for a messaging schema, describing the messages and their headers and body contents. "The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how REST APIs are described."

Once we have written our OpenAPI schema, the final task is to generate code to produce and consume messages that comply with that schema. For this, we use the OpenAPI Generator open source project. "OpenAPI Generator allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an OpenAPI Spec". It supports dozens of languages/frameworks.

	Contents of One-Day Training Course
Target Audience Software architects and senior developers tasked with efficiently creating and/or consuming REST APIs **Prerequisites** Sound understanding of at least one client programming environment and one server environment (from the OpenAPI Generator list of supported languages) Some previous experience of network programming would be useful.	**Need for REST APIs** Representational state transfer Principles of REST architecture Reliably connecting distributed pieces of functionality **Review of JSON and YAML** How we represent information on the wire JavaScript Object Notation (advanced) "YAML Ain't Markup Language" What enhancements YAML brings **Tasks To Build REST APIs** Need to consider design of API Need to specify API Need to generate client library to call API Need to integrate client library with UX Need to generate server stubs to host API Need to integrate server stubs with DB, etc Need to comprehensively test everything **Designing a REST API** What to take into consideration Functionality discoverability Identifying resources and selecting verbs Integration with HTTP methods **OpenAPI Overview** OpenAPI Initiative (Linux Foundation) The OpenAPI specification (https://github.com/OAI/OpenAPI-Specification/tree/master/versions) **OpenAPI Usage** Contract Driven API (create contract first) Handling paths Defining components Creating API documentation	**OpenAPI In Detail** Detailed look at important constructs in spec file and how best to use them Boilerplate layout and sections **OpenAPI Generator Overview** A modern open source community API Generator toolset for building REST APIs Compliant with OpenAPI Spec v3 (fork of Swagger Codegen) A set of useful tools for code generation for client libraries and server stubs e.g.: OpenAPI Generator v4.x supports Angular code generation. **OpenAPI Generator Features** Setup and tooling Integrate with build Useful command line arguments List of options and their uses Exploring OpenAPI Generator source tree **Practical REST API Creation** Security (avoiding code injection) Testing Working with CI/CD Error handling Documentation generation **Style Guide** How best to structure OpenAPI files Naming conventions Preparing for future API version evolution **Project** Building an end-to-end application involving all steps in REST API design, specification, implementation and usage

ASP.NET Core 8 REST API Development
Foundations, Creating REST APIs, Controllers, Domain, OpenAPI (Swagger), Testing, Architecture

ASP.NET Core 8 is the modern approach to developing web UI and REST API applications using .NET 8. This exciting technology richly supports OpenAPI-based REST development, test-driven development, domain models, custom URL routing schemes, flexible viewing and separation of concerns.

This course covers in-depth the foundations of ASP.NET Core 8 and how to use it to create REST APIs. Important: this course does not cover ASP.NET MVC user interface development (e.g. use of Razor) though much of what is covered here is needed for that.

ASP.NET Core 8 comes with a range of features to build REST APIs: both the HTTPS interface and what is behind it (domain model, database access via EF Core 8) + enhanced OpenAPI support.

Modern REST API developers use C# 12, .NET 8 and ASP.NET to build server-side implementations, then use OpenAPI (formerly called Swagger) to describe an API and tools such as OpenAPI-Generator to auto-generate client-side stubs (e.g. for Angular UI in the browser).

	Contents of One-Day Training Course
Target Audience REST API developers wishing to learn about the latest approach to creating REST APIs using .NET 8 technologies. **Prerequisites** Attendees are expected to be experienced C# developers with some understanding of the HTTP protocol. Attendance at our *.NET 8 CLR Programming* course or equivalent experience. No previous experience of ASP.NET required, as this course explores it from the foundations upwards.	**Overview of ASP.NET Core 8** "ASP.NET is an open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends. " [link] Goals for ASP.NET Advantages and challenges Web UI and Web API unified **Review Of REST API Pipeline** Dev setup required (Visual Studio 2022) What components are needed to deliver REST solution via ASP.NET Quick Tour of all of ASP.NET Core 8 Focus in on server-side implementation Detailed look at HTTP request pipeline **First REST API App** Walkthrough of a simple app introducing all the major features Solution layout Project template in Visual Studio 2022 ASP.NET 8 runs on .NET 8 **Important Concepts** Attributes (Produces, Route, ..) Use of async HTTP methods (POST, GET, ..) Accessing parameter information Conventions / Analyzers **.NET Generic Host** Useful for HTTP and non-HTTP traffic Structuring a host Integration with IIS and HTTP.sys **Controllers And Actions** Plays coordination role ControllerBase ApiController Action results Different results for different needs Parameterizing actions Formatting results **Domain & Data Models** Domain-driven design Tour of DDD concepts (e.g. repository) In ASP.NET context, model is a domain model (not a data model) Domain model define repository, which data model implements (e.g. EF Core 8) **Architecture** Designing ASP.NET solutions Common design problems **OpenAPI (Swagger)** Use of add-on OpenAPI capabilities with .NET 8 to create portable representation of API and documentation **Security** Web application security Using ASP.NET application services Various threats **Testing** Approaches to testing web apps Unit testing and Mocking Debugging tools (e.g. Fiddler, Postman) **Project** A demonstration of how to use ASP.NET Core for more substantial workloads

ProtoBuf And gRPC
Architecture, ProtoBuf, .proto file, Generated Code, Security, Performance, Messaging Design, Project

gRPC (https://grpc.io/) is the world's leading multi-language RPC framework and toolkit. Originally created by Google for their internal large distributed systems, gRPC is now at the heart of a thriving open source community with robust implementations for all major programming languages and runtimes.

We see gRPC as one of a number of ways of performing inter-app messaging. It is interesting to noted recently TensorFlow, which started with a gRPC interface, has added REST API support, whereas ASP.NET, which started with only allowing apps written with it to offer a REST API, now also lets them support gRPC access. gRPC is being used in many industry-leading solutions (e.g. the critical etcd component that is at the heart of Kubernetes talks gRPC to the other parts of the Kubernetes universe).

Software developers creating modern client / edge applications and cloud-based microservices would be well advised to consider using gRPC to connect them. This course aims to equip experienced developers with what the y need to know in order to quickly implement gRPC in production-quality code.

Contents of One-Day Training Course		
Target Audience Experienced software developers who need to add fast networking capabilities to the applications and microservices them are developing. **Prerequisites** (https://grpc.io/docs/) Good knowledge of one of the programming languages that support gRPC – attendees may select one of these to complete the lab exercises and project. .	**How Processes Communicate** .. on same machine or distributed IPC (interprocess communication) options Impact of corporate firewalls gRPC vs. IPC vs. REST API vs. socket - why RPC is often[not always] best choice **gRPC Overview** Overview of gRPC architecture Role of ProtoBuf .proto file Service definition **Dev Environment** What is needed on developer workstation in order to program against gRPC Library / tooling setup Auto-generating ProtoBuf details **ProtoBuf Intro** What are protocol buffers (ProtoBufs)? Supported data types Dealing with issues regarding differences between programming language data types and ProtoBuf data types Producing and consuming ProtoBufs Serializing buffers **Advanced ProtoBuf** Binary serialization Advice for buffer design and optimization Streams **gRPC Programming** Constructing a service definition Generating client and server stubs Programming details for gRPC Integration with threading and async	**Error Handling** Instrumentation and tracing Dealing with errors at protocol level and at application level Error scenarios and their responses (retry..) **Testing gRPC Apps** CI/CD and gRPC Unit testing gRPC components Role of dependency injection Integration testing **gRPC And Performance** The key selling point of gRPC is its efficiency – for high-volume messaging between software components, it is the best approach in most cases Scaling to larger number of endpoints **gRPC and Security** Authentication Encryption Denial of service **Designing Message Flows** Take time to properly design message flow Often multiple messages within a flow **gRPC Internals** Let's travel along with a packet exchange over RPC Layering HTTP/2 transport **Project** Using gRPC in a larger project – creating ProtoBuf representations of packets that needed to be exchanged and implementing messaging over gRPC

Kubernetes
How Everything Works Together, etcd, API Server, Scheduler, Controller, Kubelet, CRI, CNI, CSI, Project

Once you go beyond running a few containers on a single machine, you need to think about how to manage multiple nodes and make them work together at scale and that is where Kubernetes comes in. Kubernetes is the leading orchestration system for container clusters.

At first glance Kubernetes is confusing, because it involves multiple components running demanding / varying workloads on different nodes in a network. It needs to offer resilience in the face of all kinds of failure, very tight security, efficiency, timely delivery of application updates and many more features.

However, after we first review the big picture for Kubernetes, when we examine each of its components in turn, we see they each performs a focused task (storage config, change data, schedule a container, start a container). Each component on its own is relatively simple, and so we gradually build up a deeper understanding of how Kubernetes actually works.

This intensive course brings engineering professionals who already know about containers and microservices up to speed with the world's leading open source container orchestration platform.

	Contents of One-Day Training Course
Target Audience Developers, devops personnel and system administrators wishing to provision large clusters of containers using Kubernetes. **Prerequisites** Attendees to this course must have already attended our *Microservices and Containers* course or have similar experience.	**Big Picture Concepts** What is a container cluster, a master node, a worker node, schedulers, controllers, a container, a pod, kubelet, etc. How everything works together **Setting up a Container Cluster** Set up Kubernetes so that master talks to kubelet on each node, which manages a set of pods on that node, and each pod contains one or more containers **etcd** etcd is a highly reliable distributed database engine that optimally stores and allows observing of config data Like what you have in /etc config files but managed by daemons (hence name) RAFT Protocol **API Server** Offers a REST API to remote clients to configure and monitor a cluster Clients could be command-line interface (CLI) tools or admin consoles or scripts Writes settings into etcd Let's examine APIs on offer **Kubectl** A CLI that talks to API Server Main route for devops people to directly interact with API Server **Scheduler** Watches etcd for changes to configuration (e.g. via API Server) and makes scheduling decisions, writes these to etcd Scheduling algorithms and settings **Controllers** Multiple controller types are supplied to manage node lifetimes, replication, endpoints and account details Cloud controllers **Kubelet** Makes a worker node part of the cluster Watches settings in etcd for this node When change detected in etcd config, makes it so in what runs on worker node **Container Runtime Interface** CRI is an interface between kubelet and actual container runtime Popular options: CRI-o or cri-containerd Containers are usually run via OCI's runc **CNI** Container Network Interface (CNI) is a spec and implementation for networking functionality in the world of containers **CSI** Container Storage Interface (CSI) is a plugin architecture for storage volumes **Labels and selectors** Attaching key/value pairs to objects Selecting many objects based on labels **Kubernetes Source Tree** To build up deeper knowledge, we explore the source trees for Kubernetes and related projects, such as etcd (all written in Go) **Project** Deploying a real world application to a Kubernetes cluster

Technology / UX School

Client UI

- User Interaction Design
- HTML And DOM Using TypeScript
- CSS
- Angular 18 Fundamentals
- Advanced Angular 18
- Angular Material 18
- PWA using Angular 18 Service-Worker
- Angular 18 Internationali[z|s]ation
- NgRx
- Web Components/Angular Elements/Microfrontends

Server UI

- Node.js 22, Express And PUG UI Programming Using TypeScript

Hybrid UI

- Ionic 8

RTC

- WebRTC Using Angular

Graphics

- Web 2D Graphics Programming
- WebGPU And WGSL

Media

- Web Media Programming

Workspace

- Windows Shell Namespace Extensions

Fundamentals Of User Interaction
User Centered Design, Usability Principles, Design Process, Identifying the User, Tools, Usability Tests

Usability is an intangible software feature - more noticeable by its absence than its presence in an application. This course helps you design it into your products. User interaction design is a branch of software engineering – and needs to be approached in an organized manner. We need to get to know the user, to discover the user interaction requirements, to build the app, and then to test that we have done it correctly. This is both similar and different to how we would code up an algorithm or implement a database schema. Every workitem on a software dev team's project plan should be examined to see if it enhances usability of the app -

if not, it needs to be reconsidered. There is a need to collect as much information as possible from the user – via product demonstrations, usability testing, and after launch, analyzing the helpdesk queries. Just like advertising, 50% of software features are never used – you will need to find out which, and more importantly why – perhaps the unused features are too difficult, perhaps the user does not know they exist (inaccessible), or perhaps they are simply not needed. Designers and developers will benefit from attending this course by gaining an increasing awareness of the processes used to create incredible user-friendly apps.

	Contents of One-Day Training Course	
	User-Centric Design	**The look of the application**
	Placing the user at the center of the software design process	Font, color, etc.
		Use of certain controls
	Every developer needs to think of the user experience – not just those who directly create the user interface	Layout of forms
Target Audience		Task-centric design
This course is aimed at software designers and senior developers who need to create highly intuitive user interfaces		Icons and cursors
	Usability Design Principles	Benefits of a "minimalist" interface
	Terminology and metaphors	**Types of GUI**
	Consistency	OOUI
	Task invocation and navigation	MUI
	Functionality discovery	Handling complexity
	Continuous feedback	Handling large data sets
	Controllability	**Usability for different apps**
	Selection and activation	Database (transactions, locking, tables)
	Validation of user input	Graphics (dirty bit, perspective, selection)
	Concepts	Components (object display, activation)
	User – application communication	Networking (node selection, time delays)
Prerequisites	Norman's Model	**User Assistance Tools**
Experience of programming graphical user interfaces is needed along with an appreciation of usability issues	From CLI to GUI to CBI	Help system/online mentoring wizard
	Command-Based Interface used from multiple sources (GUI, wizards, macros, Automation, CBT/Help, undo/redo)	Context sensitive computer based training
		Error Avoidance
	Usability Design Process	Why errors happen
	Identifying the user	Engineering user errors out of applications
	Task analysis	Users never make errors-only designers do
	Story-boarding	**Usability Testing**
	Prototyping	The "five minute" user test
	Usability testing	Collecting information from users
	An iterative cycle	User interaction engineering
	Direct Manipulation	Making it part of the development project and what internal doc/models are needed.
	Direct interaction	**Usability Engineering Project**
	In-place editing	Complete walk through of how to design
	Drag and drop	user interaction for a complex project

HTML and DOM Using TypeScript
Text, sections, forms, tables, the DOM, events, the canvas and lots more inside modern browsers

The web platform is undergoing a rapid pace of innovation and its primary standards, HTML and its DOM, are central to how modern browsers work. The HTML markup pre-populates the tree and then the DOM can be used to dynamically edit the tree's content. After a quick review of the fundamentals of HTML and the DOM, this course explores many of the the new and expanded capabilities offered by HTML & DOM. These latest specs are well supported across modern web browsers from different vendors and offer an increasing range of rich functionality that significantly improve over the simple markup apps of the past.

Among the many enhancements explored in this course are the shadow DOM, the HTML canvas, better table functionality, improved event handling architecture, the idea of an HTML application with a manifest, better navigation and plenty more.

In the past developers using higher level frameworks were sometimes isolated from directly accessing HTML and the DOM, but there has been a trend in more recent frameworks to re-introduce developers to direct low-level access to the full power of HTML and the DOM - we will investigate the real benefits of this approach.

Contents of One-Day Training Course

Target Audience
Experienced developers who already know the basics of HTML (HEAD, BODY, P, DIV, etc.) and now wish to get up to speed with the latest standards for markup and programmatically editing HTML content.

Prerequisites
For the DOM part of the course, all demos and lab exercises will be in TypeScript, so attendees need to know that language.

HTML overview
WHATWG (whatwg.org) and W3C
"Living standard" vs. numbered spec
Central role of HTML's 1200-page spec
  in standardizing the web platform
Collection of specs
Metadata, elements, attributes, encoding

Tour Of Elements
Exploring common elements, some we are
  familiar with, some new to HTML5
Page lifecycle / Uses for URIs
The execution context(s)
Content models

Resources, URI and IRI
Identifying resources
Internationalization with IRI

DOM Overview
Correspondence between HTML markup
  elements and DOM tree elements (they
  are similar, but not exactly the same)
How to interact with a tree of elements
Options for parsing / serializing DOM
Role of WebIDL in defining web APIs

Sections & Grouping
Sections - body, article, section, aside,
h1..6, hgroup, header, footer, address
Grouping – p, hr, pre, blockquote, ol, ul,
menu, li, dl, dt, fig[ure|caption], main, div

Text
a, em, strong, small, s, cite, q, dfn,
 abbr, ruby, rt, rp, data, time, code,
 var, samp, sub/sup, kbd, i/b/u, mark,
bdi,bdo,span,br, wbr

Modern Tables
Evolution of HTML tables - table, caption,
colgroup, col, tbody, thead, tfoot, tr, td, th

Forms
 Submitting to the server
 form, label, input, button, select, datalist,
optgroup, option, textarea, output, progress,
meter, fieldset, legend

HTML Templates
Chunks of reusable markup
Defining and instantiating

Event Handling
 Event bubbling
 Defining and using events
 Popular DOM events
 Event listeners

Shadow DOM
Isolating elements for web components
attachshadow() and shadow root

Specialist HTML
iframe, dialog, summary, ins/del
Session history
Location

HTML Canvas
 Drawing context
2D immediate mode graphics
Other graphics and media handling
OffscreenCanvas

HTML Applications
 Application Cache and manifest
 Offline web applications
 Linking and link types
 The Markdown DSL

CSS – Cascading Style Sheets
Handling Styling, Selectors, Descriptions, Specificity, Priority, Pseudo-, CSS for …, SCSS

CSS is not a markup language (unlike HTML5) used to represent content. CSS is not a programming language (unlike JavaScript/TypeScript), used to dynamically change content at run time. Rather, CSS is a styling language used to concisely represent styling information for web content.

CSS is used to declaratively describe the initial hierarchical set of styles that are to be applied to HTML elements which later may be manipulated by TypeScript/JavaScript code. CSS offers considerable flexibility in how styles are specified and how they can be applied to elements. An important consideration is to maintain the same "look & feel" across multiple pages, Thus a site-wide "house style" is often defined. Also CSS allows the same content to be optimally rendered on output devices with differing capabilities (size, resolution, interactivity) and suitably for people with differing interaction needs.

This course looks at the role of CSS in the world of modern web development, provides a detailed tour of CSS features and explores how you can best deploy it for styling your own web solutions.

	Contents of One-Day Training Course	
Target Audience All developers wishing to gain a comprehensive understanding of the least known of the three pillars of web development (the other two being HTML and JavaScript/TypeScript). **Prerequisites** Knowledge of HTML is required along with some graphics background.	**CSS Overview** Growing set of styling standards Defining CSS rules Style document layout Declaration block Internal CSS using <style> External CSS using <link> and .css file Could be used to "render" speech, but mostly used for visual representations **CSS 1 / CSS 2 / CSS 3 / CSS 4** Evolution of standards (CSS modules) Expanding set of layered standards What is [well|less well] supported **Selector** Identifying categories of markup elements Element type Id Class Element hierarchy **Declaration Blocks** Collection of declarations in blocks Property and value (separated by colon) Representing values using custom syntax suitable for each value type **Inheritance** Importance of style ancestry Styling descendants **Pseudo-class/pseudo-element** Additional ways to select elements Styling after particular actions (e.g. click) **Combining selectors** Importance of selector order Priority	**CSS for Layout** Defining where elements appear on page Replacement for HTML tables (sometimes) Multi-columns and flexible layout **CSS for Color** Defining color scheme for elements Element color Background Ways to define color (e.g. gradient) **CSS for Font** Selecting font types for text Styling such fonts (italics, bold) Sizing fonts **CSS for Spacing** Margin Border Internal text spacing **CSS For Text** Alignment Paged media Translations **CSS For Content Handling** Web animation Transitions Generated content **SCSS** A CSS pre-processor that simplifies CSS syntax and extends its capabilities "allows you to use variables, nested rules, mixins, inline imports and more" with CSS **Project** How to use CSS in a large enterprise solution to provide flexible styling

Angular 18 Fundamentals
Tour, CLI, Building Components, Directives, Databinding, HTTP Client, Forms, Bootstrapping

Angular 18 is the most advanced web client framework available for production use. It provides a very solid foundation for development of modern web applications and is increasingly being selected by start-ups, cloud vendors and global enterprises for their large scale solutions with demanding needs and tight development time frames.

Angular is a vibrant open source project and is built by a large team from Google and elsewhere. They work to further evolve the framework and they regularly add interesting enhancements. This course covers the latest Angular 18 that can be used to build web desktop, web mobile, native desktop, native mobile and even server applications (as it has plenty of non-UI functionality).

Angular is a big project, best approached in stages by developers wishing to start building apps. Before this course app developers should learn TypeScript (Angular itself is written in TypeScript, as are most Angular apps). Then app developers should attend this course as their first contact with Angular, optionally followed by our *Advanced Angular 18* course, which looks in more detail at some specialist Angular topics.

	Contents of One-Day Training Course	
Target Audience Developers seeking to quickly get up to speed with the best web framework in the world. **Prerequisites** Developers experienced with the TypeScript language and web programming in general. No previous Angular experience required. All demos and lab exercises will be in TypeScript 5.	**Angular Framework Tour** Collection of packages that work together to deliver a wonderful web framework Overview of how it works Introduction to each major module Many parts to an Angular app **Angular CLI** Command line interface to creating, building, serving and testing Angular apps Angular CLI automates the creation of a good boilerplate source tree for your app that you can later enhance What you might like to customize (e.g. versions in package.json) CLI Prompts **Building Components** Exploring how we build components Event handling – firing and listening Input and output properties Metadata for components **Angular Template Syntax** Enhancing HTML syntax with custom directives and expressions Which HTML concepts not permitted Interpolation Expression syntax Attribute directives ngModel **Structural Directives** ngFor / ngIf ngSwitch Microsyntax ng-template / ng-container	**Databinding in-depth** Event, Property, Attribute, Class, Style, Two-Way **HTTP Client** How to use the various HTTP request types Role of in-memory-web-api for testing Use of services in Angular app architecture Asynchronous stream of events (objects) delivered to your components (RxJS) **Animation** Transitions States & Triggers **Advanced Components** Styling for Angular components Lifecycle hooks Pipes Deeper look at how you build services **Introduction to Forms** Template-driven forms Error handling Change tracking Structuring form handling code Reactive forms **Validation** Validating forms input Correctly displaying error information Built-in and custom validators Role of CSS in reflecting control status **Bootstrapping** How an Angular app bootstraps Intro to how rendering works Use of platform-browser

Advanced Angular 18
Routing, Libraries, Rendering, Platforms, DI, NgModule, Universal, DevKit, Schematics

It is in the more advanced capabilities of Angular that we see it distancing itself from simpler frameworks and results in it being more and more selected for large-scale important projects that needs a stable, powerful framework as the basis for long-term innovation.

Angular offers a well thought out architecture, its configurable platforms means alternative rendering approaches may be supported (e.g. from a web worker), dependency injection means components can be swapped in and out over time and a wonderful routing engine provides browser-side navigation for views.

Developers already familiar with using Angular to build UI apps will find this advanced course of particular interest as it comprehensively explores how to leverage the rich feature set of the Angular Framework to build more innovative applications that distinguish themselves from the competition in rendering performance and the flexibility of what they offer users.

Attendees will also benefit from this course's coverage of more specialist Angular topics, such as NgModules, Angular Universal and Angular Dev Kit (including Schematics).

Contents of One-Day Training Course	
Target Audience Angular and TypeScript developers wishing to explore more deeply how to leverage the more advanced capabilities of the Angular Framework. **Prerequisites** Attendees should have attended our *"Angular 18 Fundamentals"* course or have equivalent experience. All demos & labs are in TypeScript 5.	**Angular System Programming** Closer look at low level details of how Angular apps actually run — Use of zones in Angular — **Dependency Injection** Excellent for testing — Hierarchical DI built into Angular — Provides greater flexibility in how a well structured app can evolve into future — **Angular Router Fundamentals** Browser-based editing of address bar URI — Setting up routes using the Router Service — Router outlet — Catch-all entry — **Advanced Angular Router** State management — Handling routing events — Use of guards — **Lazy Loading** For large applications, loading everything at startup is costly — How can we postpone loading some functionality until/unless it is needed — **Advanced Angular CLI** What is a CLI workspace? — In-depth look at workspace layout — `ng add` — Build tooling — Multiple projects — **Building Angular Libraries** ng-packagr — `ng g library` — Sub-dividing large projects into libraries	**NgModule** Detailed look at what `NgModules` are and how they are used — `entryComponents` vs. `bootstrap` `imports` & `exports` — **Platforms & Rendering** Ivy renderer/compilation engine — How Ivy works and its benefits to apps — Customizing rendering — Logging rendering information — Role of platforms — **Schematics** Idea behind schematics — Creating for your own projects — Examing sample schematics projects — **Angular Language Service** "The Angular Language Service is a way to get completions, errors, hints, and navigation inside your Angular templates" — **Angular Universal** Running Angular on the server — Good for search engines (SEO) — Good for fast first load of page in browser — **Angular 18 DevKit** Set of dev tools and libraries for ecosystem — Schematics - "generators that transform an existing filesystem" -used by Angular CLI — Core, Build & Architect packages — **App Architecture** Structure needed for large Angular apps: domain model, REST API client, routing, error handling, 18n, modularity, security and lots more

Angular Material 18
Material Design, Angular Material Tour, Navigation, Layout, Popups, Datatable, Flex Layout, CDK, Project

After developers get up to speed with the fundamentals of Angular and TypeScript programming, their attention turns from the basics of creating web UIs to more substantial concerns – such as the need to create visually appealing, logically organized and easily navigable applications that responsively react to the devices used by end-users.

Such modern web applications need to be competitive in the marketplace and for this consistent styling, layout and component architecture is needed – hence the need for material design (https://material.io) and its implem-entation for Angular 18, which comes in the form of the new Angular Components repo (evolution of the Angular Material repo). Hundreds of millions of users worldwide see material design everyday when using Google Search, Gmail, Youtube and Android; hence it makes sense to adopt it for your own website too.

The three main parts of Angular Components are a well crafted set of components, the flex layout engine and CDK (for building your own components). All of these are covered in this specialist course as we explore how best to create modern web UIs that look well/work well.

Contents of One-Day Training Course		
Target Audience Developers interested in efficiently bring material design to their Angular 18 applications **Prerequisites** Experienced Angular developers with a flair for UI design. Note: This is not an introductory Angular course - so attendees must already be familiar with the Angular framework. All demos and labs will be in TypeScript, so attendees need to know TypeScript.	**Purpose of Material Design** Google's design language Style guide++ Principles of material design **Tour of Material Design** "Material Design is a unified system that combines theory, resources, and tools for crafting digital experiences." **Angular Components Overview** Implements material design for Angular Exploring its capabilities What it brings to modern UI projects Review of Angular's forms architecture **Delivering UI Capabilities** Pagination & navigation Typography Layout Useful role of schematics **Form Controls** Autocomplete, checkbox, input, radio button, select, slider, slide toggle **Navigation** Menu Sidenav (creation using schematics) Toolbar / tooltip **Layout** List, Grid, Card, Tabs Virtual Scrolling - large lists & fast UI Drag and Drop **Buttons, Indicators & Icons** Button, button toggle, chips, icons, ripple, tabs, stepper, slide-toggle Progress spinner, progress bar	**Popups And Modals** Dialog, Tooltip Snackbar **Data Table** Table and its data source Paginator and sort header Creation using schematics **Tree** Tree root and hierarchical nodes Creation using schematics Event handling **Introduction to Flex layout** Building responsive user interfaces Flexbox usage for various screen sizes The benefit of a grid and column spans Responsive API & Media query Coding custom layout How flex layout works under the hood **Architecture of CDK** Separate Component Development Kit "general building blocks for UI components decoupled from the visuals of Material Design" New foundation for Angular Material, can also be used separately **Using CDK** GitHub location: components/src/cdk Creating your own components based on CDK for a range of capabilities without too much extra development effort CDK data table **Project** Using Angular Material in a large project to see how to best utilize its rich capabilities

PWA using Angular 18 Service-Worker
PWA Ideas, AppShell, Manifest, Service Worker spec, CLI's --service-worker, Implementation, Project

Web developers have been enviously glancing over at native app developers and admiring all the shiny extra toys they have – the sometimes connected app, instant app start, app store for discoverability, notifications, etc. (of course, web developers do not forget the unique advantages they have – a ubiquitous platform, use of URLs for deep linking, avoiding version hell, etc.).

With the arrival of Progressive Web Applications (PWA), now web developers have the best of both worlds. PWA makes it easy to build web apps that run in modern browsers and behave like native apps.

A PWA is a web app that uses three key technologies – the manifest, service workers and optionally the app shell. The best way to build PWAs is to use Angular 18 & its *Service-Worker* package. The project source tree can be created as normal using Angular CLI and then run this to add PWA support: `ng add @angular/pwa`

The aim of this course to to bring Angular developers up to speed with how a PWA works, to review the underlying spec and then to dive deeply into how to implement a powerful PWA using Angular's Service-Worker package.

Contents of One-Day Training Course		
Target Audience	**PWA – What are we trying to achieve**	**Service Workers Spec**
Web developers wishing to build powerful PWA apps using Angular and its *Service-Worker* package.	All three ideas are important: - progressive - web - applications The idea of a client-side mini-proxy server and transient network connections The fact we gain native app capabilities does not mean we lose web app capabilities	Review of W3C Service Workers Spec – we explore how service workers perform Main artifacts **Angular CLI Generated Code** Add PWA using: `ng add @angular/pwa` What does it do? Add new packages: ServiceWorker & PWA angular.json/configurations/serviceWorker ServiceWorkerModule.register call
Prerequisites	**PWA Technologies**	**ngsw-config.json**
Good experience of Angular 18 programming. This is an advanced course and before attending, attendees should already be quite familiar with Angular 18 and TypeScript programming.	Manifest Service Worker AppShell Review of how these technologies work Message flows Service workers are quite distinct from web workers – not to be confused	assetGroups InstallMode (prefetch) UpdateMode Resources **Working With Service Workers** `swUpdate`: deciding on an update strategy `swPush`: service worker's push notifications Other aspects of service worker programming and configuration
	Intro to PWA with Angular	**Tooling**
	Angular and Progressive Web Apps What Angular offers to PWA app devs Exploring the Service-Worker package Angular CLI and Service-Worker apps	Google's PWA site Google LightHouse Debugging and instrumentation
	AppShell	**Internals**
	An approach to initially offer a minimalist UI that can be cached Gradually add more content Very fast rendering of first view	The source for Angular's Service-Worker package is well worth studying We also look at source for Angular CLI's @angular/pwa package
	Manifest	**Project**
	What's in a manifest.json file? Generating default Start url, scope, display, etc.	Creating an Angular application that builds on the ideas explored in this course and that scores 100 in LightHouse

Angular 18 Internationali[z|s]ation
Regional settings ng i18n, i18n Pipes, ngx-translate, Unicode, CDLR, Country Packs

Do you know the wonderful(ly complex) Japanese writing system has three kinds of symbols: kanji, hiragana & katakana. Do you know in Germany street addresses have the house number at the end rather than the beginning (so "7 Main Street" is written as "Hauptstraße 7"). Do you know in Bahrain, the currency has three subunits rather than two (BD 1.234 rather than $1.23). Even how to open a car door can be different. Do you know the social rules of formality and politeness are much more complex in many cultures compared to English-speaking lands. Do you know that 96% of the world's population do not live in the USA.

All this shows the world is a big place with fascinating differences and well worth visiting. The aim of this course is to help your Angular 18 apps travel well.

Critically important i18n aspects that Angular app developers need to consider include how to represent strings (in memory, on-screen and in data files); how to input characters in the user interface; how to display numeric, financial and date data according to local customs; how to manage dialog and other resource values and how to develop custom tools to help with localization. This course covers all these and lots more.

Contents of One-Day Training Course

Target Audience
Experienced Angular developers with an interest in preparing their Angular 18 apps for international markets

Prerequisites
This is not an introductory Angular course - so attendees must already be familiar with the Angular framework.

All demos and labs will be in TypeScript, so attendees need to know TypeScript.

No prior internationalization experience required.

PLEASE NOTE: the first half of this course covers general internationalization programming and the second half covers internationalization with Angular 18.

Overview
Importance of internationalization (i18n)
"We sell in your language/buy in our own"
i18n concepts - globalization, internationalization, localization

Review of how i18n works
Common source tree
Multiple localization assets
Helping the translators
Thinking of internationalization from the beginning – not as an afterthought
Externalizing assets we need to change

i18n Tooling
Setting up a suitable i18n dev / test bench
Need to get fully local OS installed (not just English OS with international strings)
XLF files and xliffmerge

i18n and the modern browser
How different browsers handle regional settings and language selection
Tools in the browser (e.g. dev tools)
Languages in use
Google Input Tools For Chrome
https://www.google.com/inputtools/chrome

Unicode
Character set / scripts
Text direction & text layout
Inputting complex languages

CDLR
"The Unicode CLDR (Common Locale Data Repository) provides key building blocks for software to support the world's languages, with..repository of locale data"

Angular's Built-in i18n tools
https://angular.io/guide/i18n
Overview of Angular and i18n
The new localize package in the main repo
The i18n attribute – translatable strings
ng xi18n

i18n Angular Pipes
i18n and DatePipe, CurrencyPipe, DecimalPipe and PercentPipe

Intro to ngx-translate
http://www.ngx-translate.com
This modular library provides:
- A service
- A directive
- A pipe
Good for dynamic and static content

Advanced ngx-translate
Idea of loaders
Review of provided loaders
Comparison with Angular's built-in i18n tooling

Country Packs
Often need country specific additions (e.g. for local regulations)
Building (lazily loaded) country packs for country-specific add-on functionality
Using Angular CLI's `ng g library`
Integrating country packs with main app

Project
A larger sample Angular project showing the right way and wrong ways to manage product development aimed at global markets

NgRx
Reactive + State, State Store, Side Effects, DevTools, Entity, Schematics, Architecture, Project

We use the term "state" to describe nuggets of data whose lifetime outlives that of a single call to an event handler (e.g. auth token, contents of shopping cart, custom color selection for sidenav). An Angular app is composed of a hierarchy of components. Sometimes state that is only used by a single component can be stored within that component; state shared between related components (near each other in the hierarchy) can be passed among them directly. Using services with dependency injection is also an option; but for more substantial applications with many components, managing state needs more attention. Enter NgRx ...

NgRx is a well organized suite of packages to manage application state in a RxJS observable cache. It is very popular, because it comprehensively solves the state management issue that every large Angular app ultimately will face. The three key participants in NgRx are actions, state and reducers (which literally reduce an existing state and an action to a new state). Though it can be used on the server, in the real world NgRx is mostly used on the client (on the server, data usually ends up in a database). In this detailed course we explore the world of NgRx and see how it can be of real benefit to larger Angular apps.

	Contents of One-Day Training Course

Target Audience
Developers wishing to manage client-side state in their Angular apps using NgRx
 	
Prerequisites
Experienced Angular developers building larger applications who need to more seriously consider their client-side state management architecture.
	Reactive Meets State
	What problem are we trying to solve
	The observable pattern and its uses
	Parts of RxJS that are of interest to us
	Lets look at state within an Angular app
	When to use NgRx (and when not to)
	Overview of NgRx
	Exploring the NgRx Platform
	Important packages and their interactions
	Getting it installed and running
	Adding to development environment
	Concepts
	State (and the idea of immutability)
	Actions (state changes)
	Reducers
	@nrgx/store Introduction
	`OnPush` change detection strategy
	Boilerplate code for NgRx
	Creating reducers
	Advanced @nrgx/store
	State composition
	Selectors
	ngrx-store-localstorage
	Meta-Reducers
	Meta actions & meta reducers
	Wrapping a reducer with a meta reducer
	Parameter to `StoreModule.forRoot()`
	@ngrx/effects
	What is an effects model?
	Isolating change, making pure components
	Feeding actions into state cache
	Event sourcing architecture
	`@Effect()` decorator
		@ngrx/router-store
		Combining Angular Router & @ngrx/store
		`ROUTER_NAVIGATION`
		`StoreRouterConnectingModule`
		@ngrx/entity
		What is an entity collection?
		CRUD operations on entity collections
		Uses for type-safe adapters/entity selectors
		@ngrx/store-devtools
		Redux DevTools
		Instrumentation tooling for the store
		Instrumentation options
		@ngrx/schematics
		Idea of schematics
		A scaffolding library that integrates NgRx
		  with Angular CLI
		Available blueprints
		Case Study: NgRx & Shopping Cart
		NgRx is very often used to build the
		  shopping cart feature in eCommerce apps
		Let's see how to do it properly
		NgRx Internals
		Tour of NgRx source code
		  https://github.com/ngrx/platform
		NgRx source is managed as a monorepo
		Discover how it all fits together
		Architecture Considerations
		Exploring the architectural issues that need to be considered in order to successfully leverage NgRx rich capabilities in an app
		Project
		Combining what we have learnt in the course to use in a larger NgRx project

Web Components/Angular Elements/Microfrontends
Custom Elements, Shadow DOM, NgElement, createCustomElement, App Shell, Architecture

Full stack developers have been successfully using microservices server-side for a few years and now they would like to apply that architecture client-side, an approach known as microfrontends. In some ways microfrontends are similar to microservices: a clear need to break up a monolith application, allow different parts of a large app to evolve and be deployed at their own pace, perhaps using distinct foundational technologies. In other ways, they are different: microservices can run separately in data centers on a (e.g. Kubernetes) container cluster, whereas we wish a set of microfrontends to run isolated in a web browser and yet appear to end-users to be a single integrated interactive application, with some shared capabilities. In this specialist course we first review W3C Web Components - a set of standards that allow components to be produced and consumed by different web frameworks (or different versions of the same framework). Then we look at Angular Elements, which allows the construction of web components using Angular. Then we explore microfrontends - what are they, how to best build them using Angular Elements, how to host them (app shell) & how to design large-scale web applications using them.

Contents of One-Day Training Course		

Target Audience
Experienced Angular developers working on large Angular projects who wish to compose them out of microfrontends.
 	
Prerequisites
Full stack developers with good all-round experience of Angular.
 	
Awareness of role of containers and microservices for server-side development highly relevant.
	Overview
	What are we trying to achieve?
	Dividing an app into dynamic components
	* Standards - W3C Web Components
	* Implementation - Angular Elements
	* Design approach - Microfrontends
	W3C Custom Elements
	Create your own HTML elements
	Attribute, properties, events
	New `CustomElementRegistry`
	How we can use them for microfrontends
	W3C Shadow DOM
	Shadow tree and light tree
	Angular [ViewEncapsulation.ShadowDOM]
	(note: `Native` is deprecated)
	Support Features: Slots, HTML Templates/Custom Events
	Relevant additional HTML/DOM features that modern browsers support
	Composability in Angular
	Angular dynamic components
	Lazy loading & extensibility
	Specialist use of `NgModules`
	Angular Elements Intro
	`NgElement, ngBootstrap()`
	`createCustomElement()`
	Content projection in Elements
	Advanced Angular Elements
	Build process
	Managing element lifecycle / evolution
	Loading a library containing elements
	Review of [Angular Elements' source tree]
	Importance of Render3/Ivy
		Microfrontend Architecture
		What microservices brings to server app
		How best to apply idea client-side
		Microfrontend = large (somewhat contained) slice of interactive app
		Critical to make group of them appear as as a single integrated app
		Building Microfrontends
		Structuring microfrontends
		Source layout
		Expected classes and interfaces
		Design patterns
		App Shell
		Microfrontends have to live within a shell
		Custom app shell hosts microfrontends
		What capabilities it could offer: notification – recently used-quick links-app discovery
		Microfrontends and ..
		Routing
		Internationali[z|s]ation
		Security / Styling
		Sharing widgets
		Browser Containers
		Interesting idea: containers in the browser
		[containers = namespaces + cgroups]
		How best to bring idea to browser [polyfill]
		Hosting microfrontend in browser container
		Project
		Bringing together everything covered in this course, we conclude with a review of a substantial project that uses mico-frontends, based on Angular Elements

Node.js 22 HTTPS/Express/PUG Using TypeScript
Web UI app, REST API app, Node HTTPS, Express, Generator, Routing, PUG, Project

Because of its ease of development, similarity with web client-side programming, maturity along with its expanding framework and tooling story, a Node-based approach to server-side development is very compelling. Since TypeScript is usually used client-side, it make sense to also use it server-side.

This course explores using TypeScript to develop server-side web UI and REST API applications using the latest technologies based on Node.js 22. It covers a mix of technologies that together allow developers to quickly build robust server-wide web solutions.

The Node.js HTTP(S)&HTTP/2 modules are how Node apps talk to the HTTP protocol family. The Node-based Express Application Framework is how rich web UI and web API apps can be built. It exposes powerful routing and middleware capabilities. The PUG template engine (the latest version of what used to be called Jade), transforms an HTML-like template syntax plus supplied data values into HTML which is sent to the web browser for rendering. In some ways PUG competes with Angular on the client – we contrast what they offer and investigate how to use them together in the same solution (often a sensible approach).

	Contents of One-Day Training Course	
Target Audience Web developers seeking to build rich web UI and REST API apps using the latest Node runtime, Express application framework and PUG template engine. **Prerequisites** Attendance at our *Node.js 22 Runtime Programming Using TypeScript* course or equivalent experience. All demos and lab exercises will be in TypeScript.	**Building Web UI apps** Review of server-side and client-side UI development options Role of templating and access to data Creating compelling user experiences in a Node world **REST API Apps** Review of REST API concepts Designing a REST API using Node Evolving a REST API Security issues **Node.js 22 HTTP[S] Modules** HTTP message flows using Node Use of createServer to create server server.listen and server.on syntax HTTP message content – headers, etc. Error handling considerations (error.code and Exxx values) **Introduction to Express** Application framework based on Node Main components: express(), Application, Request, Response and Router Intro to what Express calls middleware **Express Generator** Recommended file layout for Node/Express apps Express-Generator constructs boilerplate code and layout quickly Reviewing generated content **Express Routing** Converting incoming URLs and query syntax to action invocations Defining and configuring a router	Implementing an API Testing an API **Express Middleware** Additional custom steps added to processing pipeline Stack of middleware functions Helpful auxiliary packages such as path, morgan and bodyParser **PUG** Template engine based on Node & Express Intro to PUG Template syntax Intro to PUG framework - runs on server and transforms syntax + data into HTML **PUG Template Syntax** Concise representation of markup Consumes data to generate contents Loops and control flow Useful shorthand for common needs **PUG API Usage** pug.compile[|File|Client|FileClient| ClientWithDependenciesTracked] pug.render and pug.renderFile Options interface **Angular Client-side UI** Server-side UI vs. Client side Why server-wide UI is important (e.g. efficient access to server-side data) even in a world with Angular Integrating Angular on the client with server-side coding **Project** Developing an integrated project that brings together all the topics covered in the course

Ionic 8
System Model, Ionic/Core, Ionic/Angular, CLI, Stencil, JSX, Capacitor, PWA Toolkit

Ionic 8 is a complete re-imagination of how Ionic works based on W3C Web Component standards, while retaining most of the existing Ionic API used by apps.

User interface developers have for decades been creating apps by weaving together their own code with pre-built blocks of functionality – these are sometimes called widgets, custom controls or just components. With Web Components, this approach is now becoming a reality for the Web Platform also. We can now have, say, a small Ionic component consumed by a larger React component hosted by a plain JS application.

Most modern web frameworks are recently adding significant Web Component support (e.g. Angular Elements) and with v5 we see this in the Ionic world.

This course brings web developers with little or no previous Ionic experience up to speed with Ionic 8 programming. We will see how Ionic 8 is a powerful framework that enables developers use TypeScript and familiar web technologies.to build apps for mobile devices. Ionic contains a set of mobile-friendly UI components, a library to interact with native services, a CLI and a toolkit for creating progressive web apps.

	Contents of One-Day Training Course
Target Audience Web developers who wish to build either PWA or native apps for mobile devices using the latest web technologies. **Prerequisites** Experience of web development, in particular Angular. During the course we discuss the relationship between Ionic v5 and Angular (with some older versions of Ionic, use of Angular was mandatory, whereas with v5 it is optional). No previous Ionic experience needed – we start with the Ionic v5 fundamentals. All demos and labs will be in TypeScript, so attendees need to know TypeScript.	**W3C Web Components** Ideas behind this W3C set of standards Custom Elements Shadow DOM HTML Templates **Ionic 8 Overview** Developing for mobile – smaller screens, use of touch, limited device features How Ionic delivers a set of capabilities to facilitate web developers building cross platform mobile apps **Concepts** Major architectural subsystems and how they work together. Building your first Ionic 8 app The toolchain **Ionic CLI** A command-line interface to generate, serve, build and package Ionic apps Important commands and their options **Ionic 8 Core** Based on Stencil UI components Themes Utils Controllers Configuration **Ionic 8 Angular** Optional Angular integration Injectable controllers Directives for virtual Interaction with Angular routing IonicModule (an NgModule) �▸ **Stencil** Ionic Stencil is a compiler that makes Web Components Use of JSX The idea of a virtual DOM Async rendering Data binding **Capacitor** A mobile app will often need from time to time to use services of the native OS – we see Ionic Capacitor delivers capability (Capacitor takes over the role of Cordova) The Capacitor API and its usage How it works on different substrates **Ionic PWA Toolkit** Principles of progressive web apps that Ionic PWA Toolkit implements Service workers / push notifications Configuration / routing **Ionic 8 Internals** Tour of Ionic 8 source code https://github.com/ionic-team Explore how the various projects mesh together and can be extended **Architecture Considerations** How to elegantly design mobile apps Handling differences among device types Issues to pay attention to **Project** Bringing together all the ideas discussed in the course into a larger project to create a mobile app that will be competitive in the app stores

WebRTC Using Angular
Protocols, API Tour, Sessions, SDP, PeerConnection, DataChannel, Media/Codecs, NAT, Security, Ivy, Project

Real time communication (RTC) is intended for person-to-person live communication. WebRTC is built into modern standards-compliant web browsers, mobile devices and IoT devices. WebRTC implements RTC without needing browser extensions or plug-ins and browsers supporting it are already widely deployed. Forms of communication WebRTC supports include audio, video and data exchange (e.g. app data or docs).

WebRTC is the basis for RTC within many interactive apps that are widely used. What is less known is that WebRTC is a programming platform, and apps running in the browser & on mobile/IoT devices can leverage it. Developers striving to add unique features to their apps would be well advised to consider WebRTC, as once an understanding of how it works has been gained, it is not that time consuming to program. The results are quite powerful and much appreciated by users.

This course starts with an introduction to WebRTC, then looks at the protocols and API standards in detail, then looks at how to program it using TypeScript, paying close attention to everthing app developers need to know, both on the client and the server.

	Contents of One-Day Training Course	
Target Audience Experienced Angular web application developers who require a deeper understanding of WebRTC and how to use it within their own web applications. **Prerequisites** All demo code and lab exercises uses TypeScript, so web programming experience using that is required, as is good all-round foundational networking knowledge. No previous WebRTC experience required. Attendees must be experienced Angular developers.	**WebRTC Intro** What it is trying to achieve - a community, a set of standard protocols, a set of standard APIs, a foundation upon which to build apps and an open source project **Architecture** Major components for audio, video, data How transport works; role of codecs Identity and security (e.g. DTLS) How to use with TypeScript **Foundational IETF Protocols** RTP /SRTP: Realtime Transport Protocol RTCP/SRTCP: RTP Control Protocol SDP: Session Description Protocol SCTP: Stream Control Transmission Future: QUIC and WebRTC? **W3C Standards** W3C is working on many WebRTC stds Review of WebRTC 1.0: Real-time Comms Between Browsers, Media Capture and Streams and WebRTC's Statistics **Connectivity Establishment** Interactive Connectivity Establishment How to use ICE/SDP to programmatically connect with remote party [JSEP] Detailed look at RTCPeerConnection Works from DOM thread, not web worker **Data Channel Programming** Exchanging app data over WebRTC links RTCDataChannel: creation and use Using send() from Typescript Typings already defined in lib.dom.d.ts (which Angular CLI adds to tsconfig.json)	**MediaStream** MediaStream/MediaStreamTrack interfaces Codec selection (look at AV1 and OPUS) **WebRTC Media API** Passing track info to remote parties RTCRtp[Sender|Receiver|Transceiver] Encoding / transmission / processing **WebRTC on the Server** Role of signaling (custom to server app) Your server app exposes REST API MCU, SFU, gateways, ... **WebRTC and STUN/TURN** Issues with some networks (NAT/firewall) Navigating NATs with WebRTC (STUN) Relays using TURN (e.g. JANUS) **WebRTC And Security** How to identify/locate participants Securing comms links (DTLS, secure RTP) Regulation: comms, GDPR, police, .. **Application Issues** Developer environment setup for WebRTC Debugging (chrome://webrtc-internals/) Error handling with WebRTC **Reference Architecture** Review of how "Ivy" works in Angular Investigating "Ivy" custom renderers What is needed to add "Ivy" over WebRTC Reference architecture exploring how UI of an Angular client app can be shared via WebRTC's data channel **Project** Attendees will extend above project to add shared interactivity+handle user events

Web 2D Graphics Programming
HTML Canvas, SVG, Web Animation, Web Fonts, Image Element, Conversions, Coordinates, Files

This focused course explores all aspects of 2D graphics programming on the web. There are many 2D graphics options available to the modern web app developer; and they need to know which to choose for different sets of requirements.

This course covers in detail how to render onscreen & offscreen in 2D, create visual effects, download/upload image content, capture and animate visuals and how to handle fonts. We also look at user input and how to correctly respond to events (e.g. mouse moves) and how to handle screen refresh and window resizing.

Specific technologies covered include the HTML canvas element and its contexts, Canvas 2D graphical APIs, structured graphics with SVG, fonts with WOFF2, web animations using CSS and lots more.

We also investigate 2D graphics programming as part of larger application development and how to correctly structure how the graphical code interacts with other parts of the larger application – an issue often ignored until it is too late in the development process.

Contents of One-Day Training Course

Target Audience
Experienced web developers seeking to programmatically create 2D graphics.

Prerequisites
Some background of graphical programming is mandatory, as is web programming experience.

All demo and lab code will be using Angular and TypeScript, so experience of both of these is needed.

Note: This course covers 2D only – it does not cover 3D graphics on the web. The W3C is completing a new 3D standard for the web called WebGPU.and we offer a separate dedicated course on this.

W3C Standards
The W3C offer a number of specs in the area of 2D graphics
Web developers have plenty of options but need to choose carefully depending on requirements
Tour of what is available
Introduction to each programming model

HTML Canvas Context
The <canvas> html element
Where a canvas can be contained and what it itself can contain
Canvas attributes (width, height, ..)
The idea of a rendering context
`CanvasRenderingContext2D`

Canvas Primitives
Graphical primitives: draw line, point, rect
Styles and strokes
Transform, scale, rotate, translate
Composing, path, text handling

Canvas Advanced
Offscreen bitmaps
`HtmlCanvasElement.OffscreenCanvas`
Programmatically extracting canvas data using `toBlob` and `toDataURL`
`ImageBitmapRenderingContext`

HTML Image
Images and the rest of a HTML page
`HTMLImageElement`: src, & srcset
JPEG and PNG formats
ALT and accessibility
Handling images on a web page
Interacting with images via advanced CSS

SVG Intro
Scalable Vector Graphics is just that XML-based
Unlike canvas (no file format), SVG is retained graphics with stream presentation
`SVGImageElement`
SVG in standalone file / HTML embedded
Drawing primitives and coordinate system

SVG Advanced
Styling with style attributes
Fills, strokes, effects, widths
Paths and line segments
Filtering
Advanced effects

HTML Web Animation
Relationship to CSS Transitions, CSS Animations and SVG
Configuring animations
Keyframes
Timelines

Angular's Web Animation
Use of CSS animatable properties
Enabling Angular's animation package
Transitions and timing in TypeScript
Triggers

Web Open Font Format (WOFF)
Typography on the web
WOFF - W3C spec to represent fonts
https://fonts.google.com/

Project
Building a larger project that integrates all of the 2D graphics programming capabilities explored in this course

WebGPU And WGSL
Modern 3D on the web; Spec; Adapters/Devices, Buffers, Textures, Layouts, Commands, Pipelines

WebGPU is the rapidly maturing next-generation 3D specification for the standard web platform. WebGPU has been created by the W3C's *GPU For the Web Community Group* with the active participation of all major browser vendors – Google, Mozilla, Microsoft and Apple. Most modern client devices (including modern mobile devices) now have some form of 3D hardware and correctly exposing this to web apps in a secure and high-performant way is the goal. WebGPU works the way modern graphics is intended to be programmed – with a Web IDL-defined API running on the CPU and a shader running directly on the GPU.

Shaders are written using the brand new WGSL (WebGPU Shader Language). WebGPU is suitable for both graphics- and compute-workloads. A significant subset of the graphics functionality of modern GPUs is also of great interest for non-graphics (e.g. machine learning) programming (e.g. heavy-duty matrix multiplication). The intent of this code-intensive course is to bring already experienced web application developers up to speed with all things WebGPU and have them create their own web applications using WebGPU that lerverages the available underlying 3D hardware as much as possible.

Contents of One-Day Training Course		

Target Audience
 Web application developers and 3D framework developers seeking to leverage the latest 3D technology on the standard web.
 	
Prerequisites
Developers experienced with web programming and 3D graphics.
 	
Sample code and lab exercises will be in TypeScript, so knowledge of this language is required.
	Review of WebGPU Standard
	What does it offer: the most advanced compute and 3D rendering pipeline
	Clearly focused on securely exposing modern graphics hardware to web apps
	What does it not cover: shading, AR/VR, 2D, audio, input/touch:see other specs
	WebGPU API Technical Tour
	Command Buffers / GPU buffers
	Pipelines and shaders
	Devices and adapters
	Textures / resources / Rendering
	GPUAdapter and GPUDevice
	`navigator.gpu` (worker / main thread)
	GPUAdapter (the GPU hardware)
	Access via call to `gpu.requestAdapter`
	Config: `GPURequestAdapterOptions`
	(power setting for now, future:expandable)
	GPUDevice: context to interact with GPU
	Access via `adapter.requestDevice`
	May be multiple "devices" per adapter
	Has "create" function for all constructs
	Buffer
	Managed blocks of memory
	`createBuffer[|Mapped|MappedAsync]`
	Mapped means accessible as ArrayBuffer
	GPUBufferDescriptor (size + usage)
	Usage flags – e.g. `map_read, copy_dst`
	Textures
	Create textures using `createTexture`
	`GPUTextureDescriptor`
	Create views: `GPUTexture.createView`
	`Review of texture formats`
		Samplers
		Role of samplers
		`createSampler`
		`GPUSamplerDescriptor`
		Addressing modes / filters
		Layout
		"A GPUBindGroupLayout defines the interface between a set of resources bound in a GPUBindGroup and their accessibility in shader stages." [link]
		`createBindGroup[|Layout]`
		`GPUBindGroup[|Layout]Descriptor`
		`createPipelineLayout`
		`GPUPipelineLayoutDescriptor`
		ShaderModule & Pipeline
		Shaders & WebGPU
		`createShaderModule`
		`GPUShaderModuleDescriptor`
		`create[Compute|Render]Pipeline`
		`GPU[Compute|Render]PipelineDescriptor`
		WGSL-WebGPU Shader Language
		Close relationship to SPIR-V
		Suitable for many targets
		Will be universally supported
		WGSL data types and syntax
		Encoders
		`createCommandEncoder`
		`GPUCommandEncoderDescriptor`
		`createRenderBundleEncoder`
		`GPURenderBundleEncoderDescriptor`
		Development with WebGPU
		Exploring development issues when creating web applications using WebGPU
		App architecture suggestions

Web Media Programming
Video/audio in the web browser, Media Stream, Web Audio API, Audio Graph, Nodes, Codecs, Project

A modern user experience requires content in a multitude of formats, including audio and video. As measured by bandwidth, audiovisual media content is the most popular type of content on the web. Media plays a very important when showing products at trade shows, at sales presentations, use of a deployed product, training, support and more. Web application developers need a clear understanding of how media on the web works, how to program against the exposed media APIs, how to choose codecs and how to incorporate media functionality as one (important) part of a larger user-facing web application.

The goal of this wide-ranging course is to help experience d web application developers become web media programmers. Media functionality can in added to new & existing web applications in a number of way, often with a surprisingly small amount of code (the hard work has already been done inside the browser itself).

This course also covers some of the more recent additions to web media programming, such as the new Web Audio spec, the AV1 codec, and the Media Recorder API; along with the recent evolution of the existing web media specs.

	Contents of One-Day Training Course	
Target Audience Web application developers who wish to quickly add rich media capabilities to their existing web applications. **Prerequisites** Experienced web application developers with at least a high-level understanding of media concepts from a programming viewpoint. All sample code will be in TypeScript, so attendees need to know how to program in that language.	**Web Media Ecosystem** Overview of media on the web - web servers, web browsers, proxies Importance of CDNs Media and .. RTC, apps, offline, ++ **Protocols for Streaming** How server can send media to browser Overview of protocols – RTSP, RTMP, RTP/RTCP and more Security and media **HTML Element For Media** HTML \<video\> tag and attributes HTML \<audio\> tag and attributes (autoplay, loop, muted, preload, ..) `HTML[Video|Audio]Element` and their parent, `HTMLMediaElement` MIME types for media **MediaStream** How media streams work in the browser How tracks work `MediaStream` interface `MediaStreamTrack` interface **Capture** Capturing the user's media devices (with permission) The `getUserMedia()` call Detailed configuration options **Media Recording API** The `MediaRecorder` object The `dataavailable` event How to programmatically record audiovisual media streams	**Web Audio Overview** Purpose and features of Web Audio A very comprehensive audio architecture `AudioContext` Channels What is an audio graph? Use of nodes for source/filter/compression **Advanced Web Audio** Context options Spatial (`PannerNode` for 3D positioning) Audio worklets Handling the ended event Splitting and merging Audio generation **Offline Web Audio** `OfflineAudioContext` and audio buffers Audio in the background **Codecs** Web application developers are unlikely to write their own codec for production use (maybe do it for curiosity) - unless huge effort invested, custom code will not be competitive with mainstream options However, web devs do need high level understanding of codecs usage and choice We focus on AV1 video codec and OPUS audio codec, often used together **Project** We create a larger web project with audio and video functionality as one part of a multifaceted experience and investigate how the interaction between the media & non-media sides can best be structured

Windows Shell Namespace Extensions
Extend Windows Explorer with IShellFolder2, IEnumIDList, IContextMenu3, IDataObject

On the Windows Desktop, shell extensions enable independent software vendors to extend the Windows Shell and its namespace with their own functionality. The Windows shell namespace is an extensible hierarchical collection of naming and other info related to directories, files, printers and networks. The Windows Shell, including the Desktop, File Explorer and the common file open/save dialogs used by most apps, provides extensible graphical browsers / editors for the shell namespace. All users of Windows are familiar with the File Explorer tool, which lets them browse the content of the hard disk/LAN/Control Panel.

There are a number of custom sources of hierarchical data that would nicely extend the "system" namespace. By programmatically extending File Explorer we may provide integrated browsing capabilities for these.

Shell namespace extensions are an ideal solution where there is a requirement to list items and attributes in a hierarchical manner, or when data files should be managed on a medium other than file systems (e.g. in a database, cloud backup or on remote devices). Other forms of shell extensions provide access to extended functionality such as context menus or property pages.

	Contents of One-Day Training Course
Target Audience This training course targets experienced C++ and COM developers who want to create shell namespace extensions. **Prerequisites** Detailed experience of C++ and COM development. (Note: Shell extensions should not be written in .NET).	**Shell Namespace** A PC's rooted shell namespace is a hierarchical collection of: • System components such as file systems, printers, the control panel • Optional system extensions • Third-party namespace extensions **COM DLL Refresher** Quick review and comparison of how to create COM DLLs in C++ using a variety of techniques **Explorer & Common File Dialogs** System tools for browsing the shell namespace. Writing your own browsers **Architecture** Examination of how a shell namespace extension may be structured. Registry settings. Creating a simple non-foldering example. The IShellFolder2 interface **Foldering** Develop single level foldering example Item ID Lists & the IEnumIDList interface Cloning and non-cloning varieties of multi-level foldering / Node providers Important registry settings **GUI** Extending Explorer's toolbar, menubar and status bar / IExtractIcon Examination of how a shell namespace extension should be structured Develop single level foldering example Item ID Lists & the IEnumIDList interface

Data Transfer
How data transfer works
The IDataObject, IDropSource and IDropTarget interfaces

Data Flow
The crux of the problem is how to enable data to flow to all parts of the code in a namespace extension which requires it -
• IEnumIDList for foldering
• IDataObject for drag & drop and for the common file dialogs

Emulating File Systems
To a certain extent, though not fully, a namespace extension may emulate a file system
We examine what does and does not work
Importance of SFGAO_FILESYSTEM attr
Example of editing files on a remote FTP server
Special requirements for Microsoft Office file dialogs

Other Types of Shell Extensions
Context menu handler
• Drag and Drop Handler
• Icon Handler
• Property Sheet
Designing a custom context menu handler

Project
Full sample project showing how to create all parts of a custom Windows Shell namespace extension;
Architecture deep dive

Operations / DevOps

Toolchain
- GIT
- TypeScript/Node/Angular Toolchain
- C/C++ Toolchain

Process
- Agile Development Process

C/C++ Toolchain
Understand, Organize, Build, Write, Document, Test, Optimize, Verify, Debug, Package, Explore, Deploy

Lots of things have to happen correctly in order to efficiently transform source code on developers' computers into commercial digital products in use by customers around the world.

After writing the source, the code has to be built, unit tested, documented for developers, stored in repositories, optimized, debugged, the performance monitored, the source style has to be verified, libraries and components packaged & applications deployed. For C/C++ developers, a toolchain is needed with appropriate tools for each of these tasks.

The C/C++ toolchain often gets less attention than the languages themselves or high-profile libraries (such as STL) but it is vital for highly productive developers to have a deep understanding of the C/C++ toolchain and what it can offer. As projects get larger and there is pressure to deliver updates in shorter time frames but with higher quality expectations, successfully leveraging the C/C++ toolchain and all its rich capabilities makes all the difference.

This course explores a range of useful developer tools that when used together achieves just that.

	Contents of One-Day Training Course	
Target Audience Software engineers who wish to more fully explore the toolchain options available for C/C++ projects **Prerequisites** Experience of C and/or C++ programming. At least some experience of the C/C++ toolchain.	**Tour of C/C++ Toolchain** Range of tools needs for high-productivity high-quality C/C++ programming **Intro to CMake** CMake is the popular cross-platform C/C++ build system Relationship to native build systems Defining the build process with CMake Handling the build & source directory tree **VIM** Options for editing code across platforms General use of VIM VIM for C/C++ source editing **Intro to Unit Testing** Benefits of unit tests & overview Designing tests and automatic evaluation Intro to Google C++ Testing Framework **Advanced Unit Testing** More detailed look at Google C++ Testing Framework and how it can be successfully used in large C/C++ projects Mocking, code coverage, performance **Code Documentation** Doxygen auto-generating documentation Professional finish **Lint, etc.** Making sure code complies with styling and other requirements Use of lint and other specialist tools **Larger Source Trees** Organizing large C/C++ source trees Header files, libs, main	**C/C++ Compilers** The three major compilation toolkits are GCC, LLVM and Visual C++ compiler plus there are many smaller toolkits **LLVM** Tour of LLVM project CLang Optimizer Standard library **Language Service for C/C++** Using LLVM to build C/C++ code Command-line options Compilation passes **Debugging** Native debugger Extracting run time information Symbol management **Package Management** Unlike other languages, C/C++ does not have popular package management systems (e.g. no C/C++ equivalent of npm or nuget) We look at what are the packaging options available to C/C++ apps (e.g. chocolatey) **Deployment** Approaches to packaging C/C++ apps for different platforms Deciding where & how to place assets **Real-world Toolchain Usage** Exploring the toolchain setup for a large open-source C/C++ project **Project** Correctly setting up toolchain usage for our own enterprise projects

TypeScript/Node/Angular Toolchain
Understand, Organize, Build, Write, Document, Test, Optimize, Verify, Debug, Package, Explore, Deploy

Lots of things have to happen correctly in order to efficiently transform source code on developers' computers into commercial digital products in use by customers around the world.

After writing the source, the code has to be transpiled, unit tested, documented for developers, stored in repositories, optimized, debugged, the performance monitored, the source style has to be verified, libraries and components packaged, minified & applications deployed. For TypeScript developers, a toolchain is needed with appropriate tools for each of these tasks.

The TypeScript/Node/Angular toolchain often gets less attention than the language/runtimes themselves but it is vital for highly productive developers to have a deep understanding of the toolchain and what it can offer. As projects get larger and there is pressure to deliver updates in shorter time frames but with higher quality expectations, successfully leveraging the TypeScript/Node/Angular toolchain and all its rich capabilities makes all the difference.

This course explores a range of useful developer tools that when used together achieves just that.

	Contents of One-Day Training Course

Target Audience
Software engineers who wish to more fully explore the toolchain options available for TypeScript projects for Node and/or Angular.

Prerequisites
Experience of TypeScript programming for Node and/or Angular.

Toolchain Tour
Range of tools needs for high-productivity high-quality TypeScript/Node/Angular programming
Node's Inspector Protocol

TSC – TypeScript Transpiler
Tsc and its options
More advanced uses for configuration file

TypeScript Language Service
Written in TypeScript, an external process supplying a range of language services
How to call from non-TypeScript clients
Navigating your TypeScript AST in code

Visual Studio Code
Code editor for various languages
Use of Visual Studio Code for TypeScript programming, both Node and Angular

TypeDoc Code Documentation
TypeDoc auto-generating documentation
Professional finish

Eslint / TSLint
These tools are used to make sure code complies with styling/maintainability

Angular Package Format
The Angular Package Format is a well-specified layout used for Angular packages
Third-party libraries should follow layout

ng-packagr
Role of ng-packagr in the build pipeline
Extending Angular with libraries

Angular Augury
In-depth debugging of Angular content
Chrome integration

ts-node
Node-based execution environment for TypeScript code

Gulp.ts
Gulp is a very popular task runner
We explore how to write gulp files in TypeScript (requires ts-node)
Defining the build process using gulp.ts
Handling the build & source directory tree

WebPack
Role of module bundler
Dependency graph and bundles

NPM & Yarn
Creating packages using Node Package manager (NPM) and Yarn
Publishing to NPM
Adding extra features to package (e.g. CI)

Jasmine & Karma
What to test; Test syntax in Jasmine
Karma is an excellent test runner
Executing tests using karma; config.js

Real-world Toolchain Usage
Exploring the toolchain setup for the main Angular project itself – a very large open source TypeScript that is widely deployed

Project
Correctly setting up toolchain usage for our own enterprise projects

Agile Software Development
Methodology, Values, Principles, User Stories, Pairing, Individual/Team Dynamics, Test First, Refactoring

Agile software development is a highly productive strategy for organizing project teams. It allows software to be quickly written for today's needs and that can easily evolve to keep up with an every-changing business environment. It exploits the best of individual and team creative efforts. Agile shuns the heavy "ceremony" of documentation-rich rigid software development processes; and instead advocates a code-centric approach with just enough documentation to genuinely satisfy all stakeholders' needs. This approach is more satisfying for developers and is more finely aligned with customers' true requirements. It results in much richer contributions from all project participants. It advocates short iterations that fulfills the "release early and release often" aspiration. Teams are based on a flat organizational structure with a high-degree of face-to-face communication with a "product owner". It treats the software development business as an iterative subscription model, rather than a once-off event. It is a simpler approach to writing software – and easier for a team to use. In today's highly competitive environment, whichever software team can write quality code fastest wins. This course is ideally suited to dev teams making the agile move to gain competitive advantage.

	Contents of One-Day Training Course	
	Efficient Light Methodology	**Agile Projects**
	Current state of play in s/w projects	Agile project management
	Need for new approach to programming	Iterations and releases
	Families of agile methodologies and	**Requirements via User Stories**
	selecting best techniques from each	What is a user story and how to create one?
Target Audience	"An ecosystem that ships software"	Certain number completed each iteration
All members of software project teams – managers, architects & programmers - and the product owner.	**Values**	**Design**
	Individuals and interactions over processes and tools	Architecture with a little 'a'
		UML as a sketch
	Working software over comprehensive documentation	Design today for today; refactor tomorrow
	Custom collaboration over contract negotiation	**Test-First**
		Written by programmers (unit tests) & customers (acceptance tests)
	Responding to change over following a plan	Putting the concept into practice
Prerequisites	**Principles**	**Pair Programming**
Good understanding of software development issues	Satisfying customers, welcoming changes, delivering working software, work together, projects by motivated individuals, face-to-face communications, progress measured by running code, sustainable development, quality matters, simplicity, self-organizing teams, reflection	"A driver and a navigator" (think rally car racing) - not "a driver and a passenger"
		Duties of the navigator
Significant project experience (either as a specifier or an implementer) .		Continuous code review
	Individual & Team Dynamics	**Refactoring**
	High communication levels	What happens when software matures?
	Shared knowledge spaces	"Improve the design after the software has been written"
	Benefits of higher-skilled developers	Business Issues
	Handling a creative team	Fixed price/fixed-scope contracts, responsibilities, outsourcing, etc.
	Collaborative & competitive	
	Design Documentation	**Tools**
	Doc strategy – succinct yet sufficient	Continuous Integration/Delivery tooling
	Documentation for all project stakeholders	Unit Test tools
	Useful templates for project documents	Refactoring tools
		Sample Project
		Exploration of the use of agile processes in a case study development project

Operations / Identity

- Fundamentals of Security
- Angular Security and Authentication
- GDPR & CCPA – A Developer's Viewpoint

Fundamentals Of Security
Security Concepts, Crypto, Certs, Identity, Trust, Attack Patterns, ISO2700x, Reviews, Processes

The entire software user base - specifiers, management, users, developers, administrators – all demand security and all have a role in delivering it. The goal of this course is to teach participants a common core set of fundamentals that is the first step in achieving this. Security should be treated as part of the expected skill set of every software professional. They need a fundamental understanding of security issues, before considering how to address them in the apps they develop and deploy (or better, as an integral part of the design). Security considerations must be part of software decision making, though they should not overwhelm it. Most software pros already have some awareness of security issues – this course builds on this basic knowledge and ensures the entire dev and infrastructure teams have a heightened and consistent appreciation of security concepts, along with a deep understanding of the core security standards.

This course focuses on the fundamental concepts and standards behind security. It is independent of any operating system or software environments Those attending will be well placed afterwards to think about optimum implementation strategies for their platforms.

Contents of One-Day Training Course		
Target Audience Software developers and IT. professionals who need a good grounding in all the important security concepts **Prerequisites** Experience of working on software projects, including development, deployment and ongoing service provision. No previous security programming or infrastructure experience is required, though any such knowledge would be beneficial.	**Security Services** Message integrity Authentication Non-repudiation Proof of submission/delivery Confidentiality Privacy Anonymity **Security Concepts** Network authentication, authorization, auditing, ciphers, key exchange, hashing, salting, least privilege, default lockdown-mode, canonicalization, leaks, buffer overflows, attacks (dictionary,mitm) **Cryptography** Symmetric and asymmetric crypto Latest crypto standards AES and SHA-3 Elliptic Curve Cryptography Comparison of performance & robustness Problems with older specs [des/md5] **Digital Certificates** Public Key Infrastructure (PKI) Revocation and CRL Attributes, certificate fields **Identity** Identity management Identity and federation Limiting dispersal of identity **Trust Services** Offloading work to trusted third parties Whom to trust, how, and to what extent? Trust server	**Common Attack Patterns** Social engineering Web app attacks and insider attacks Human factors **ISO 2700x** International standards for identifying, documenting and countering threats The proposed ISO 2700x series Purpose of ISO 27001 Information Security Management Systems **Security Reviews** Conducting security reviews Security threats – from inside and outside Building a threat model **A Security Development Process** Integral part of how we write software Best practices as part of dev process Ongoing influence **A Security Infrastructure Process** Security policy in the enterprise Secure deployment and operations Advisories – CERT, vendor-specific **Design Patterns for Security** How to correctly design security features into your software systems **Security and …** Storage, backups, networking, WiFi, user interface, identity, kernel, etc. **Project** Designing a secure programmable infrastructure for a sample system

Angular Security and Authentication
CSP, Contexts, Sanitizers, Schema, XSS, CSRF, CORS, XSSI, Authentication Workflow, Web-Authn

There are many design choices web developers make every day that can positively/negatively impact web app security. Security is not a task just to be left to security experts (though having them on the team is certainly a good idea). No, every web developer needs a strong grounding in both web security in general and the security of the web framework they use in particular. This course supplies both for Angular app developers. We start with a thorough review of general browser security and then proceed to see how Angular can help in building secure web applications, including exploring in detail how to build authentication.

Angular has a compelling security story, responding well to potential attack vectors. By default an Angular CLI-generated app is very secure. As code is added, security settings can be carefully adjusted as needed.

Authentication is one of the most complex and yet most important aspect of any substantial Angular application. By breaking it into manageable chunks, attendees will appreciate how a well organized authentication workflow should be, and see how to build this inside an Angular application (using NgRx to store the auth token). We also look at authorization and auditing.

	Contents of One-Day Training Course	
Target Audience Angular / TypeScript developers wishing to gain a deeper understanding of how web security in general and security within an Angular app in particular work. Also those who need to implement authentication within an Angular app. **Prerequisites** This is an advanced course and as prerequisites attendees must have an understanding of security fundamentals and general experience of Angular development	**Overview of Browser Security** TLS / SSL Input validation / output encoding Client-side security Open Web Application Security Project **Content Security Policy (CSP)** CSP feature tour Directives Policy definition Integration with other specifications **Strict Transport Security** "Defines a mechanism enabling web sites to declare themselves accessible only via secure connections" - RFC **Web Cryptography** Running cryptographic algorithms inside a browser (AES, RSA, HMAC, SHA, etc.) A W3C recommendation **Overview of Angular Security** Angular's security best practice XSS CSRF Security contexts & sanitizers "Security risk" marking in doc **Cross-Site Scripting (XSS)** Idea of malicious code-centric Protecting the DOM Tackling XSS in Angular **Sanitizers** Types of Angular security contexts Role of sanitizers Custom sanitizers Review of schema for security definitions	**Cross-Site Request Forgery (XSRF)** What browser app needs to prevent XSRF Angular HttpClient and XSRF **CORS** Cross-Origin Request and CORS request A server-side feature / Enabling CORs **Cross-Site Script Inclusion (XSSI)** JSON APIs and security Preventing execution of JSON responses **Dev Tools & Security** Review of Chrome Devtools' Security tab Lighthouse Security audits **Angular Authentication** Designing auth workflow for Angular app Login / logout UI & status UI API calls for auth and retrieving token Storing JWT auth token / supply to APIs AuthGuard for routing **Implementing Authentication** Important decisions / security implications Public vs. secure routing targets Using NgRx to manage the auth token Two-factor authentication (using Twilio) Extending with authorization and auditing **Web Authentication API** FIDO and W3C have released web-authn Web-authn use by secure Angular app Bringing together the ideas covered in this course to design and build a platform

GDPR & CCPA – A Developer's Viewpoint
Privacy Primer, Privacy & Software Platforms, Implementing Consent, SAR, Breach, Compliance

One of the most significant yet understated achievements of the European Union in practically helping its 500 million citizens is the GDPR – General Data Protection Regulation (EU) 2016/679. The upcoming California Consumer Privacy Act (CCPA), sometimes called the "American GDPR", is substantially similar to the GDPR but has some differences. Though somewhat bureaucreatic, when you understand how GDPR and CCPA work, they are actually quite a sensible approach. If you were asked to invent your own data protection framework that genuinely protects citizen data privacy rights, you probably would end up with something close to GDPR/CCPA. Global companies are being encouraged to implement GDPR on a world-wide basis (with suitable local adjustments), because it is good for their customers and to be ready for the equivalent rules that are likely to be adopted by governments around the world over the next few years.

This specialist course for software developers explores how the architecture of their global software platforms can be adjusted to accommodate implementing GDPR / CCPA and similar. It focuses in on how to implement software features needed to deliver robust data privacy.

	Contents of One-Day Training Course	
Target Audience Experienced software engineers who need to implement GDPR-/CCPA-related functionality in the software platforms they are developing. IMPORTANT: This course is not legal advice. This course is written and presented by a software engineer, not a lawyer. To get a complete all-round explanation of all aspects of GDPR/CCPA and to get legal advice on how their organizations should implement these, attendees also separately need to consult with qualified lawyers licensed for the jurisdiction(s) relevant to their project(s). **Prerequisites** Good all-round experience of software development, at a CTO, software architect or senior software developer level.	**Privacy and Data Protection** What are we trying to protect? What are the possible rights that could be conferred on citizens/companies? The idea of a data protection framework and how it might work Categories of data / heightened protection for some (e.g. personal health data) Introducing GDPR is like when recycling arrived - a bit of a change at first, soon became embedded in how we behave **A layman's GDPR/CCPA Intro** What is the GDPR/CCPA/similar? Organizations have responsibilities Individuals have rights What are data controllers, data processors and data subjects? CCPA vs. GDPR Lawful reasons for processing their data Adequacy Decision & international **GDPR/CCPA & Software Platforms** What are important features of GDPR/CCPA for cloud and enterprise software platforms? GDPR/CCPA as one feature of the wider corporate software infrastructure **Privacy Model for Software Platforms** Recommend a corporate privacy model, with input from engineering, ops, sales, .. Want one clear global set of features / procedures that encompass global privacy requirements and can adapt in future	**Categories** Personal data stored needs to be identified and assigned a category-how to implement Some categories (e.g. health details) are required to have higher protection-how? **Consent: Asking For/Recording** One lawful reason for storing/processing an individual's data is they have consented How software platforms can ask use for consent+record the granted/denied consent **SAR Handling (access request)** Data subject may ask for copy of their data Software platforms should be adjusted to electronically accept SARs and respond How do we know individual making SAR is really the data subject? What data to collect (and what format) How to return result via software platform **Handling a Data Breach** When data breach occurs, inform data controller + data subject (if at risk) **Handling an Audit** An audit can check for GDPR compliance – how dev/DevOps best prepare for audit **Specialist Issues** Personal data appearing in log files What happens when employee joins/leaves What happens with new customer Paying attention to data retention policy **GDPR Software Project** We examine the practical steps needed to GDPR-enable a software platform