Week 1 Curriculum

* Anatomy of Code (Language, Compiler, Runtime, Platform)
* Environment Setup (IDE, Code Editor, Version Control, Chat)
* Basic Topics (Core C#, Program Structure, Testing, Logging)
* .NET Building Blocks (Framework, Standard, Core, Project, Solution, Assembly, Library, Application)
* Common Language Runtime (BCL, CIL, CLI, CLR, CTS, JIT, VES)
* Runtime Environment (Garbage Collection, Managed, Unmanaged)
* Datatypes (Reference, Value)
* Access Modifiers (Internal, Private, Protected, Public)
* Extended Modifiers (Abstract, Const, New, Override, Partial, Readonly, Sealed, Static, Virtual)
  + Partial – why? Reason is if there is some kind of code generation, you don’t want your code to be deleted multiple times
  + Abstract –
  + Const –
  + New –
  + Override –
  + Readonly –
  + Sealed –
  + Static –
  + Virtual -
* Class (Constructor, Field, Method, Property, Reference Type)
* Struct (Constructor, Field, Method, Property, Value Type)
* Interface (Method, Property)
* Enum (Collection)
* Semantic Code (DRY, Comments-Inline, Comments-XML, KISS, Separation Of Concerns)
* Object Oriented Programming (Abstraction, Encapsulation, Inheritance, Polymorphism)
* Variance (As, Boxing, Casting, Is, Out, Ref, TypeOf)
* Collections (Array, List, Data Structures, Dictionary, Generics)
* Serialization (File I/O, Regular Expressions, Serializer-JSON, Serializer-Text, Serializer-XML)
* Exception Handling (Catch, Custom Exceptions, Finally, Throw, Try)
* Test Driven Development (Code-Coverage, Unit-Testing)
* Application Debugging (Breakpoints, Debugger, Logging)
  + Logging - Helps with debugging, especially when it’s a problem out in production.
    - Technology
      * (Nlog(XML config. and Serilog(code))
    - Syntax??
    - Logging hierarchy
      1. Fatal
      2. Error
      3. Warning
      4. Info
      5. Debug
      6. Trace
* SOLID (Single-Responsibility, Open-Closed, Liskov-Substitution, Interface-Segregation, Dependency-Inversion)
  + Single- Responsibility principle – while each class above all and also each method, each interface and each assembly etc. should have exactly 1 responsibility? What is a responsibility? Should have 1 reason to change
  + Open-Closed Principle – code should be open to extension, closed to modification(its an attitude to how we evolve our code base) not every version I shouldn’t redesign everything. My code should be able to be extended with new features later
  + Liskov-Substitution principle – anywhere in code it should be possible to replace an object with instances of their subtypes, without affecting the correctness of the behavior(I have a shape parent class: any code that accepts a shape should accept another shape)
  + Interface Segregation principle – prefer many granular interfaces rather than few broad purpose interfaces(if im writing a method of my own and I need to depend on some object that implements data access)
  + Dependency Inversion Principle – Code should depend on abstractions (interfaces) not concrete classes.
* Design Patterns (Creational-Singleton, Creational-Factory)
* Delegates (Action, Event, Function, Lambda, LINQ, Predicate)
* Multithreading (Async-Await, Task, Thread)
* Git (Add, Commit, Log, Pull, Push, Status)