

Introduction to group

- UiPath batch 2 -> RPA (Robotic Process Automation)
 - Week 1-4
 - .Net
 - C#
 - SQL
 - Asp.Net (MVC, SOAP, RESTful)
 - Front End (HTML, CSS, JS, VB)
 - Week 5-8
 - UiPath Foundation -> Studio
 - UiPath Advanced -> ReFramework and Orchectrator
 - UiPath New features and advanced training -> Hyperautomation and ELK
 - Week 9-10
 - P3, Portfolios, Client Interviews, and showcase
- Training Operations
 - Evaluations
 - Quizzes
 - small ones daily[5-10 questions]
 - big ones [60 questions 45 minutes] on Mondays
 - 1 on 1s on Mondays
 - QC on Mondays
 - panels - week 9
 - Projects:
 - P0 - individual project covers C#
 - P1- individual project covers C# and SQL
 - P2- group project
 - P3- batch project working in groups

.NET Architectural Components

.NET Core

.Net => Platform => to create apps (CUI, GUI, microservices, cloud services, IOT, game development, ML)

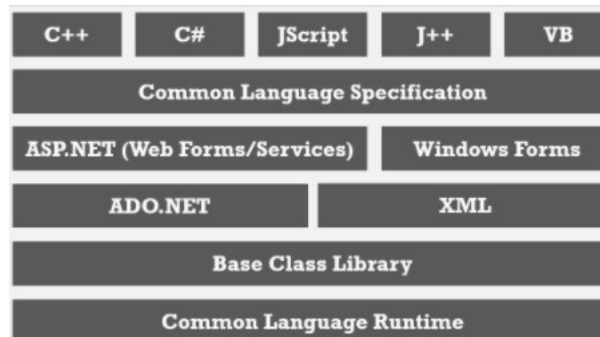
2002 -> .Net framework 1.0

2020 -> .Net

	1		
Framework	Core (2016)	Xamarin	Mono
(Windows)	platform independent	Phone Apps	(MacOS)

- .NET Standard
 - specification of .NET APIs to make up a uniform set of contracts to compile code against
 - enables use across different .NET implementations allowing the code to run everywhere
 - target framework
 - if code targets a version of Standard it can run on any .NET implementations supporting that version

- versions are additive all APIs from previous versions are incorporated by later versions
- versions are immutable
- reference assemblies distributed with NuGet packages
- NETStandard.Library metapackage references complete set of NuGet packages defining .NET Standard



- bottom up
 - Common Language Runtime (CLR) handles code execution
 - Base Class Library contains "runtime libraries" supporting common functions such as:
 - file reading and writing
 - XML document manipulation
 - exception handling
 - application globalization
 - network communication
 - threading and reflection
 - ADO.NET (library) and XML support tasks related to data access, parsing, manipulation, and generating XML
 - ASP.NET and Windows forms build robust web applications and standard Windows applications as well as develop and consume web services
 - Common Language Specification (CLS) facilitates interoperability between languages
 - defines reasonable subset of Common Type System (CTS)
 - shared data type serving as great role in cross-language integration
 - flexibility makes many languages adapt to .NET platform
- .NET Core
 - since 2016 cross-platform implementation of .NET
 - designed to handle server and cloud workloads at scale
 - Components of .Net Core SDK
 - dotnet CLI (Common Language Infrastructure)
 - dotnet libraries (standard libraries)
 - runtime/Common Language Runtime (CLR)
- UWP - Universal Windows Platform
 - implementation of .NET used for building, touch-enabled Windows apps and software
 - designed to unify different devices
 - provides
 - centralized app store
 - execution environment (AppContainer)

- set of Windows APIs
- apps can be written in C++, C#, Visual Basic, and JavaScript
- .NET Runtimes
 - runtime is execution environment for managed program
 - examples include:
 - Common Language Runtime (CLR) for .NET Framework
 - Core Common Language Runtime (CoreCLR) for .NETCore
 - .NET Native for Universal Windows Platform
 - Mono runtime for Xamarin.iOS, Xamarin.Android, Xamarin.Mac, and Mono desktop framework
- ASP.NET Core
 - cross-platform, high-performance, open-source framework to build modern, cloud-based, internet connected applications
 - Benefits
 - integrated testing
 - develop and run on Windows, macOS, and Linux
 - support for hosting Remote Procedure Call (RPC) services using gRPC
 - cloud-ready, environment-based configuration system
 - build-in dependency injection
 - lightweight, modular HTTP request pipeline
 - ability to host on Docker, IIS, Kestrel and others
 - integrates with client-side frameworks and libraries

Common Language Runtime (CLR)

- .NET Framework consists of CLR and .NET Framework class library
- CLR is foundation for .NET Framework
 - manages and runs code
 - provides services like memory management, remoting, type enforcement (through CTS), and security
- Benefits
 - cross-language integration
 - cross-language exception handling
 - enhanced security
 - versioning and deployment support
 - simplified model for component interaction
 - debugging and profiling services
- .NET Class Libraries
 - BCL (Base Class Library)
 - foundation for C# runtime library and one of the CLI standard libraries
 - provides types that represent built-in CLI data types, basic file access, collections, custom attributes, formatting, security attributes, I/O streams, string manipulation, etc.
 - class library
 - comprehensive, object-oriented collection of reusable types that can be used to develop apps
- .NET CoreFX
 - fork of BCL
 - foundational class library for .NET Core
 - defines types for primitives, collections, file systems, console, JSON, XML, async, etc.

- makes up .NET Core BCL
 - Managed Code
 - managed by CLR at runtime
 - CLR knows what code is doing and can manage it
 - CLR provides memory management (GC), security boundaries, type safety, etc.
 - written in high-level language that can be run on top of .NET
 - compiled into Intermediate Language code which CLR compiles and executes
 - manages Just-In-Time compiling of code from IL to machine code that can be run on a CPU
 - Unmanaged Code
 - runs outside of CLR
 - .NET Framework promotes interaction with COM components, COM+ services, external type libraries, and many OS services
 - examples include: COM components, ActiveX interfaces, Windows API functions
 - IDisposable Interface
 - Garbage Collector (GC) has no knowledge of unmanaged resources (open files and streams)
 - provides a method for releasing unmanaged resources
 - call `IDisposable.Dispose` implementation when finished using it
 - Using Block
 - if language supports construct (like using statement in C#) can be used without explicitly calling `IDisposable.Dispose`
 - ensures `.Dispose` is called even if exception occurs within using block
 - can be achieved by putting object inside try block and calling `.Dispose` in finally block
 - block is expanded to try/catch block at compile time
 - Using Block and IDisposable
 - provides convenient syntax ensuring correct use of `IDisposable` objects
-

Anatomy of code (language compiler, runtime, platform)

- Procedural Programming
 - programming paradigm derived from "structured" programming based on concept of procedure call
 - procedures (routines, subroutines, functions, methods) contains series of computational steps to be carried out
 - methods can be called at any point in program's execution by other methods or by itself (recursion)
- C# Program Structure
 - key organizational concepts are programs, namespaces, types, members, and assemblies
 - consist of one or more source files
 - programs declare namespaces
 - namespaces contain types (classes/interfaces)
 - types contain members (Fields, methods, properties, events)
 - Hello World program
 - starts with using directive referencing system namespace containing console class, I/O, Collections, and others
 - namespaces provide hierarchical means of organizing C# programs and libraries; contain types and even other namespaces
 - using directive allows use of all type members of namespace
 - static method `Main` serves as entry point of program
- Generic Host

- ASP.NET Core templates create .NET Core Generic Host (HostBuilder)
- Host is object encapsulating apps resources (DI, Logging, Configuration)
 - allows lifetime management over app as it controls startup and shutdown
- host calls IHostedService.StartAsync on each implementation of IHostedService in DI container
- typically configured, build, and run by code in Program.cs
- Main method:
 - calls CreateHostBuilder method to
 - create and configure a builder object
 - calls build and run methods on builder object
- have default builder settings like:
 - setting content root to path returned by GetCurrentDirectory
 - enables scope validation and dependency validation where environment is development
- C# Structure
 - Data Types
 - 2 types of variable types are supported
 - Value Types
 - built-in primitive data types
 - char, int, and float
 - along with user-defined types declared with struct
 - directly contain their data
 - Reference Types
 - classes and other complex data types constructed from primitive types
 - contain reference to location in memory where data is held
 - Expressions
 - constructed from operands and operators
 - operands are what operators act upon (literals, fields, local variables, expressions)
 - precedence of operators controls order in which individual operators evaluated
 - Statements
 - actions of any program are expressed using statements
 - C# uses various types
 - block: consists of list of statements written between {}
 - Declaration: used to create variables and constants
 - Expression: statements used to evaluate actions and take on values
 - includes method invocations, object allocations using new, compound assignment operators, and await expressions among others
 - Selection: statements used for "flow-control"
 - select one of possible statements for execution based on value (if and switch)
 - Iteration: statements include the while, do, for, and foreach statements; repeat until condition is met
 - Jump: statements used to transfer control; contains break, continue, goto, throw, return, and yield
 - Methods
 - code block containing series of statements
 - program calls method and required arguments
 - all commands are executed within a method
 - Main method is entry point for applications
 - declared in class or struct by specifying method signature:

- access level and modifiers are optional
 - all signatures must include return value, method name, and method parameters
 - Method Invocation
 - 2 types
 - Instance methods: require object to be instantiated to be called
 - Static Methods: can be called without instantiating object
 - Classes
 - most fundamental type
 - data structure combining state (fields) and actions (methods and other functions) in a single unit
 - provides template for instances of class known as objects
 - created by using class declarations
 - start with header specifying: attributes and modifiers, name, base class(if given), and interfaces implemented by class
 - header followed by body consisting of list of member declarations
-

Common Language Infrastructure (CLI)

- part of .NET strategy enabling application program written in any programming language to be run on any operating system using a "common" runtime program rather than language-specific one
- Common Type System (CTS)
 - used by CLR to enforce strict typing and code verification
 - ensure all compilers generate managed code conforming to CTS
 - describes all data types and all related constructs supported by CLR
 - details how they must be represented in .NET metadata format
 - specifies how entities can interact with each other
 - managed code can consume other managed types and instances while enforcing type fidelity and type safety
 - provides library of basic primitive types to be used in application development
 - defines 2 main types that should be supported (reference and value)
 - defines several categories of types each with specific semantics and usage
 - defines all other properties of types (access modifiers, valid type members, how inheritance, and overloading works, etc.)
- Common Language Specification (CLS)
 - .NET has specified commonalities required to enable full interoperability between languages
 - subset of CTS so all rules apply
 - defines set of rules and restrictions every language must follow when running under .NET Framework
 - provides recipe for any language implemented on top of .NET on what it must support
- Common Intermediate Language (CIL)
 - intermediate language binary instruction set defined within CLI
 - instructions are executed by Common Language Runtime
 - languages targeting CLI compile to CIL
 - runtimes Just-In-Time(JIT) compile CIL instructions into native code
 - JIT (Just-In-Time Compiler)
 - involves compilation of program at runtime
 - assumes some code might never be called

- converts CIL as needed during execution and stores resulting code in memory to make it accessible for subsequent calls in context of process instead of converting everything no matter what
 - VES (Virtual Execution System)
 - run-time system of CLI providing environment for executing managed code where CIL instruction set can be executed
 - CLR is .NET Framework's implementation of a VES providing direct support for set of data types
 - defines:
 - hypothetical machine with associated machine model and state
 - set of control flow constructs
 - exception handling model
-

Git Essentials

- Version Control System
 - software/system that allows management of changes/developments made in coding project
- Git
 - version control system (specifically distributed version control system DVCS)
 - records changes made to project and maintains history tree with state of project at savepoint
- Repository
 - where code is stored
 - local is on your machine
 - remote is online (GitHub)
- Commit
 - version of code
 - how you create save points that you can access and be able to revert back if something goes wrong
- Branch
 - separate lines of development
 - good to have at least 2 branches where main has latest stable version and feature branches that can be merged with master branch once they work
- Staging
 - used to group files you choose to include in commit
 - if changes don't work you can revert to last code that worked
- Basic Git Commands
 - git init
 - initializes git repository in folder
 - creates empty git repo
 - git clone <link to remote repo>
 - creates local clone of remote repo
 - git add filename/.
 - adds file or files to staging
 - git status
 - checks what files are in staging
 - displays state of working directory and staging area
 - git commit -m 'message'
 - commits changes to repository
 - must include 'message'

- git push
 - pushes changes to remote repository
- git pull
 - used to get updates on changes made to remote depository being tracked
- .gitignore
 - important file in repository
 - lists all files to be ignored (not tracked), files would be pushed to remote repository
 - commonly build output