Doubts + points	Solution	Example
Bonjour	Bonjour is Apple's implementation of zero-configuration networking (zeroconf), a group of technologies that includes service discovery, address assignment, and hostname resolution. Bonjour locates devices such as printers, other computers, and the services that those devices offer on a local network using multicast Domain Name System (mDNS) service records.	
	Bonjour provides a general method to discover services on a local area network.	
	Zero-configuration networking (zeroconf) is a set of technologies that automatically creates a usable computer network based on the Internet Protocol Suite (TCP/IP) when computers or network peripherals are interconnected. It does not require manual operator intervention or special configuration servers. Zeroconf is built on three core technologies: automatic assignment of numeric network addresses for networked devices, automatic distribution and resolution of computer hostnames, and automatic location of network services, such as printing devices. Without zeroconf, a network administrator must set up network services, such as Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS), or configure each computer's network settings manually.	
FPT	The FTP protocol exchanges data using two separate channels known as the command channel and data channel.	
	The command channel typically runs on server port 21 and is responsible for accepting client connections and handling the exchange of simple commands between an FTP client and server. The USER and PASS commands used for authenticating an FTP user are examples of commands that are exchanged on the command channel. The command channel remains open until the client sends the QUIT command to disconnect, or the server forcibly disconnects the client due to inactivity or other reason.	
	The data channel, runs using on-demand temporary ports listening on the server (passive mode) or on the client (active mode) and is responsible for exchanging data in the form of directory listings and file transfers. The LIST, STOR and RETR commands used for getting a server directory listing, uploading a file and downloading a file are examples of commands (sent using the command channel) that open a data channel. Unlike the command channel which remains open during the entire FTP session, the data channel is closed once the transfer of data is complete. In order to handle concurrent file transfers or directory listings a range of data channel ports must be used.	
	Using FTP both the command and data channels are unencrypted. Any data sent over these channels can be intercepted and read. One common exploit that takes advantage of this particular vulnerability is the man-in-the-middle attack using ARP poisoning and a packet sniffer.	
SFTP	SFTP is actually based on the SSH (Secure Shell) protocol which is best known for it's use in providing secure access to shell accounts on remote servers.	
.SqlProject	We can create the replica of Database and it in Visual studio as a separate project. This process helps to keep databsase in line with code while deployment.	Need Detail study
Maven		
Team city		
Git		
SpcsFlow		
Dependency Injection		

Concepts	Discription	Example	
Managed Code	Code that targets the CLR is known as managed code.	Managed code is something which uses services provided by CLR. Code will concetrate on business logic and rest of the services will be provided by CLR. The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.	
Unmanaged Code	Code that does not targets the CLR is known as managed code.	Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low - level functions of the operating system. Background compatibility with code of VB, ASP and COM are examples of unmanaged code.	
	Unmanaged code is executed with help of wrapper classes. Wrapper classes are of two types: *CCW (COM Callable Wrapper) and RCW (Runtime Callable Wrapper)*.		https://docs.microsoft.com/en-us/dotnet/framework/interop/com-wrappers
.Net framework consists of	Two Components: 1.Common Language Runtime 2.Base Class Library		
Common Language Runtime responsible	Running of code • Memory management • Compilation of code • Provides garbage collection, error handling • Code access security for semi-trusted code*<Explore>*	*<add more features>*	
Base Class Library Resposible for	It provides base for our application development so that we dont require to develop base. For example, Frameworks provides API to read data from file, We don't need to do coding for that.		
Dot net Architecture	Languages -- > compilation of each language specific code will be done by language specific compiler and converted into IL(Intermidiated language) and then IL will be converted by JIT Compiler into Native language.		
Common Language Specification	To enable full interoperability scenarios, all objects that are created in code must rely on some commonality in the languages that are consuming them (are their callers). Since there are numerous different languages, .NET platform has specified those commonalities in something called the Common Language Specification (CLS).	*<Add more>*	
Common Type System	Common Type System (CTS) is a standard that specifies how type definitions and specific values of types are represented in computer memory.	*<Add more>*	https://docs.microsoft.com/en-us/dotnet/standard/base-types/common-type-system
CTS is in charge of	1. Establish a framework for cross-language execution. 2. Provide an object-oriented model to support implementing various languages on . NET platform. 3. Define a set of rules that all languages must follow when it comes to working with types. 4. Provide a library that contains the basic primitive types that are used in application development (such as, Boolean, Byte, Char etc.)1.		https://docs.microsoft.com/en-us/dotnet/standard/language-independence-and-language-independent-components
What is SOAP	Web Services provide mechanism for programs to communicate over internet using SOAP (Simple Object Access Protocol).	*<add example of soap request>*	
name space	it is a collection of classes. Its more used for logical organization of your classes		
DLL	Dynamic Link Library		
Assembly and type of it	an assembly is a compiled code library 1.process assemblies (EXE) and 2.library assemblies (DLL) which will be in IL Form and executed by JIT and converted into Native language at runtime.		
Pointers are missing in C#.	To maintain type safety and security, C# does not support pointer arithmetic, by default. However, by using the unsafe keyword, you can define an unsafe context in which pointers can be used. Unsafe operations such as direct memory manipulation.		
TYPES : Value Type : int bool char long,enum, structs			
Reference Type:Classes, Interfaces, Arrays, Delegates			
Storage capacity	1 Byte : char, unsigned char, signed char 2 Byte: short, unsigned short 4 Byte : int, unsigned int,long, unsigned long,float 8 byte :double,long double		

Boxing and unboxing	Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit.	int i = 123;
		// The following line boxes i.
		object o = i;
		o = 123;
		i = (int)o; // unboxing	

Nullable Types	Nullable types represent value-type variables that can be assigned the value of null. You cannot create a nullable type based on a reference type: Reference types already support the null value. A nullable type can represent the normal range of values for its underlying value type, plus an additional null value.	int? num = null;	
Implicitly Typed Local Variables	If the variable is declare as type "var" Restriction : 1.The declarator must include an initializer. 2.The initializer must be an expression. The initializer cannot be an object or collection initializer by itself, but it can be a new expression that includes an object or collection initializer. 3.The compile-time type of the initializer expression cannot be the null type. 4.If the local variable declaration includes multiple declarators, the initializers must all have the same compile-time type.	var num = 10; "var" can and should be used with LINQ queries that return custom collections. It can be used for defining the variable that holds the result of the query. It can also be used within a "foreach" loop as an iterator for cycling through the results, like this. This keyword is also used when we create anonymous objects	
Types of Constructor:	1.Default Constructor :Default Constructor will get automatically created and invoked, if constructor is not Specified in the class and assign the instance variables with their default values. 2.Static constructor: This is similar to static method. It must be parameter less and must not have an access modifier (private or public).	Default Constructor if not created in code, Automatically created by CLR used to initialize the class members. Static constructor will be called once when the first time class object will be created. Used to initializze the varible that requires to initializze once.	
Construcoter execution orders.	1.Derived static fields 2.Derived static constructor 3.Derived instance fields 4.Base static fields 5.Base static constructor 6.Base instance fields 7.Base instance constructor 8.Derived instance constructor	For Example refer this : www.csharp411.com/c-object-initialization/	

There are four kinds of parameters: out, ref, params and value.	Value :Passing the argument as value.
	value is a default parameter
	Ref:Passing the argument reference not value
	Out:The out keyword causes arguments to be passed by reference.
	Ref vs Out :This is like the ref keyword, except that ref requires that the variable be initialized before it is passed. To use an out parameter, both the method definition and the calling method must explicitly use the out keyword. out requires that variable to be initiallized in called method.For example:
	params:
		VALUE: Method(val); //Calling like this
		static void Method(int i)
		{
		//Method body - chaingin the parameter value
		}
		Ref : Method(ref val); //Calling like this
		static void Method(ref int i)
		{
		//Method body - chaingin the parameter value
		}
		Out : Method(out val); //Calling like this
		static void Method(out int i)
		{
		//Method body - chaingin the parameter value
		}	

Param :	Using "Params", the arguments passed to a method are changed by the compiler to elements in a temporary array, and this array is then used to retrieve the method.	1. It should be the last argument in method
		2. Only one is allowed.
		
		public static void ADDParameters(params int[] arguemnts)
		{
		}	

Method Overloading	Single method name but the type of parameters or the number of the patameters or the order of the type is different within the same class.		

Concepts	Discription	Example
Types Of Class Accessibility	• Public: Access is not restricted. • Private: Access is limited to the containing type. • Protected: Access is limited to the containing class or types derived from the containing class. • Internal: Access is limited to the current assembly. • Protected internal: Access is limited to the current assembly or types derived from the containing class.	
Default Access Specifier	Class and Struct has the internal as default modifier. Enum and interface has the Public as default modifier. Methods,Fields,and properties has the Private as default modifier.	

Property	They can validate data before allowing a change.
	– They can transparently expose data on a class where that data is actually retrieved from some other source, such as a database.
	– They can take an action when data is changed, such as raising an event, or changing the value of other fields.
		public int MyPropert {
		get { ..}
		 set{}
		}

Indexer	Indexers permit instances of a class or struct to be indexed in the same way as arrays.
	You are working on a class that has a collection of some sort, but you want the class to appear to users (consumers of the class) as if it is a collection.
	
	The best example is the DataRow class in ADO.NET. If you want to get the value of the fifth cell of a DataRow, you can either use DataRow.Item[4] or DataRow[4].
	So data row has implemented indexer.
		public string this[int pos]
		{
		get
		{return myData[pos];}
		set {myData[pos] = value}
		}

Inheritance	Inheritance is also called 'is a' relationship	
	1.If we are creating the instace of parent class,we will acess to the parent members only. 2.If we are creating the instace of child class,we will acess to the parent members and child members. 3.If we are creating the instace of child class and assigning to the parent ,we will acess to the parent members only. 4.We can not assign the parent obj to child class.	

New keyword use	• It is possible for a derived class to define a member that has the same name as a member in its base class.
	• When this happens, the member in the base class is hidden within the derived class.
	• Even though this is not technically an error in C#, the compiler issues a warning message.
	• If you intended to hide a base class member purposely, then to prevent this warning, the derived class member must be preceded by the new keyword.
		public class BaseC
		{
		   public static int x = 55;
		}
		public class DerivedC : BaseC
		{
		    new public static int x = 100;
		}

	IF we don't want to override the method in the derive class but want to keep the same name as parent class then we can delcare method as New keyword.	public new void method() { … }
	If we want to access base class property or method from derive class, we can use base keyword	
Method Overriding	1.If we are creating the obj of parent class and calling the method which is override in the derive class,Base method will be called. 2.If we are creating the obj of child class and calling the method which is override in the derive class,derive method will be called. 3.If we are creating the obj of child class and assigning to the parent class and calling the method through parent class which is override in the derive class,derive method will be called.	Virtual and override keyword
	If we have method in base class and with the same signature we have the method in the child class with out having overide and virtual keyword,and if we call method by super class refernce (assigned by child class), It will call base class method.It will show compile time warning that these methods are hiding(derive class's method).	
Abstract class and method	Method without the method body will be declared as abstact method and class containing atleast one abstract method called as abstarc class. An abstract class is the one that cannot be instantiated It may contain abstract and non-abstract function members It cannot be sealed. abstact class's child class either should have abstact method or concrete method. abstract class may have constructor	
Sealed keyword	• Sealed class can not be inheritate. • All structs are implicitly sealed. • Sealed method cant be override.	
difference between read only and const	http://www.dotnet-tricks.com/Tutorial/csharp/FU4N141113-Difference-Between-Constant-and-ReadOnly-and-Static.html	
Interface	• Interface members are implicitly public abstract (virtual). • Interface members must not be static. • A class can inherit from a single base class, but can implement multiple interfaces. • A struct cannot inherit from any type, but can implement multiple interfaces. • Implemented interface methods must not be declared as override. • Implemented interface methods can be declared as virtual or abstract	
Difference between interface and abstract	An Abstract class doesn't provide full abstraction but an interface does provide full abstraction. Using Abstract we can not achieve multiple inheritance but using an Interface we can achieve multiple inheritance. We can not declare a member field in an Interface but can delcare property in interface. But property needs to define in derived class. -Doubt We can not use any access modifier i.e. public , private , protected , internal etc. because within an interface by default everything is public. An Interface member cannot be defined using the keyword static, virtual, abstract or sealed.	

Explicit Interface implementation	We can implement methods by adding interace name as suffics to methods.	public class Person : IDal
		{
		       public string FirstName { get; set; }
		       public string LastName { get; set; }
		
		void IDal.Add()
		       {
		          throw new NotImplementedException();
		       }
		
		       void IDal.Update()
		       {
		          throw new NotImplementedException();
		       }
		}

	"Explicit" interface should be used when your concrete class abstraction does not include the interface abstraction	*<Explain>*
Default Method implementation :	Any one of the interface method can have method implemenatation with out appending interface name.	
Method hiding :	we can have the same methods in child class with the new keyword.	

Extension method	Allow the addition of methods to an existing class outside the class definition.
	Extension methods **cannot** be used to override existing methods.
	Extension methods are a special kind of static method
	Extension methods are defined as static methods but are called by using instance method syntax
		public static int WordCount(this String str)
		{
		          return str.Split(new char[] { ' ', '.', '?' },
		                         StringSplitOptions.RemoveEmptyEntries).Length;
		}

Extension method restriction	Extended method should be static. Class containing extended method should be static. "this" keyword will be in the parameter section of the method. You can call the method on the type of the first parameter.	
Object Initializers?	An object initializer is used to assign values to an object fields or properties when the object is created.	Customer c = new Customer() { Name = "Maria Anders", City = "Berlin" }; Customer c = new Customer(1) { Name = "Maria Anders", City = "Berlin" };
Anonymous Types	Set property values into an object without writing a class definition The resulting class has no usable name • The class name is generated by the compiler • The created class inherits from Object • The result is an 'anonymous' type that is not available at the source code level.	var person = new { Name = "John Doe", Age = 33 };
When to use Anonymous Types	•Need a temporary object to hold related data •Don't need methods •When there is a need for different set of properties for each declaration •When there is a need to change the order of the properties for each declaration.	

Concepts	Discription	Example	
When not to use Anonymous Types	When Not to use Anonymous Types •There is a need to define a method •There is a need to define another variable. •There is a need to share data across methods		
	Two kind of exceptions : 1.SystemException 2.ApplicationException		
Collection	No need to set the size at the time of the declaration.CLR will take care of it.		
	ArrayList,Stack,Queue,HashTable	hashtable will sort the data.	
Generics Collection	To provide stronger compile-time type checking and reduce type casts, Stack<T>,HashTable<K,V>,List<T>	*<Code Walkthrough of TM>*	
	1. Generic types to maximize code reuse, type safety, and performance 2. The most common use of generics is to create collection classes 3. Generic classes may be constrained to enable access to methods on particular data types 4. Information on the types that are used in a generic data type may be obtained at run-time by using reflection	Generics helps to define the template for defining the type. We dont need to define different different types based on specific primitive types. Once we create the generics class, it will automatically accomodate type at the time of declaration.	
Iterator,Ienumarable,Enumaratore Object and stuff.			
	The foreach statement repeats a group of embedded statements for each element in an array or an object collection that implements the System.Collections.IEnumerable or System.Collections.Generic.IEnumerable<T> interface. IEnumerable interface force to implement		

Constraints on Type Parameters	where T: struct The type argument must be a value type. Any value type except Nullable can be specified.
	where T : class The type argument must be a reference type; this applies also to any class, interface, delegate, or array type.
	where T : new() The type argument must have a public parameterless constructor. When used together with other constraints, the new() constraint must be specified last.
	where T : <base class name> The type argument must be or derive from the specified base class.
	where T : <interface name> The type argument must be or implement the specified interface. Multiple interface constraints can be specified. The constraining interface can also be generic.
	where T : U The type argument supplied for T must be or derive from the argument supplied for U.
		public class GenericList<T> where T : Employee
		{
		.....
		}	

	Generic class should have restriction of type IComparable Interface. So that Any type parameter must implement IComparable Interface so that generic method can be provided which can use compare method.		

	Default value for Type parameter in class is : The solution is to use the default keyword, which will return null for reference types and zero for numeric value types	public T GetLast()
		{
		T temp = default(T);
		}	

Attributes	Attributes provide a way of associating information with code in a declarative way. They can also provide a reusable element that can be applied to a variety of targets.	public class MySpecialAttribute : Attribute
		{
		}	

yield keyword	1.When you use the yield keyword in a statement, you indicate that the method, operator, or get accessor in which it appears is an iterator.
	2.forms of the yield statement :
	1.yield return <expression>;
	2.yield break;
	
	3.You consume an iterator method by using a foreach statement or LINQ query. Each iteration of the foreach loop calls the iterator method. When a yield return statement is reached in the iterator method, expression is returned, and the current location in code is retained. Execution is restarted from that location the next time that the iterator function is called.
	4.You can use a yield break statement to end the iteration.
	5.The declaration of an iterator must meet the following requirements:
	1.The return type must be IEnumerable, IEnumerable<T>, IEnumerator, or IEnumerator<T>.
	2.The declaration can't have any ref or out parameters.
		public static IEnumerable Power(int number, int exponent)
		{
		int counter = 0;
		int result = 1;
		while (counter++ < exponent)
		{
		Console.WriteLine(" resultbefore retun {0} ", result);
		result = result * number;
		yield return result;
		Console.WriteLine(" result after retun {0} ", result);
		}
		}
		static void Main()
		{
		foreach (int i in Power(2, 8))
		{
		Console.WriteLine("power result {0} ", i);
		}
		}	

Delegate	A delegate is a(type safe function pointer) type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance. Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs.	delegate double MyDelegate(double x); // Declaring a
		//delegate
		static void DemoDelegates()
		{
		// Creating & instantiating a delegate a delegate instance
		MyDelegate delInstance = new MyDelegate(Math.Sin);
		// Invoking a Method attached to a delagate
		double x = delInstance(1.0);
		}
			Delegate is type safe function pointer.delegate will point to the fuction which mentioned in the contruction.

Use of Delegates	– Use events in your program. – Improve overall performance by calling certain methods asynchronously. – Or even invoke methods with one call.	EventHandler is a system defined Delegate. When we are assigning method name to the Click(or to any other property) internally it create the object of the EventHandlare Delegate so when perticular event will occur,methd will be called.	

	If we want to pass the method as a perameter,delegate is the only solution.	class Program
		{
		public delegate bool FindEmployeeDelegate(Employee emp);
		static void Main(string[] args)
		{
		List<Employee> lst = new List<Employee>() { new Employee() { ID = 103, Name = "Nimesh" }, new Employee() { ID = 104, Name = "Nikita" } };
		FindEmpoyee(lst, x => x.ID == 103);
		Console.ReadLine();
		}
		public static void FindEmpoyee(List<Employee> emplist, FindEmployeeDelegate findEmployeeDelegate)
		{
		foreach (Employee item in emplist)
		{
		if (findEmployeeDelegate(item))
		{
		Console.WriteLine("Do something for these" + item.ID);
		}
		}
		}
		}
		public class Employee
		{
		public int ID { get; set; }
		public string Name { get; set; }
		}	

Generic Delegate	It means we will not specify the type of input parameter or return type at the time of delegate declaration.	delegate T GenericDelegate<T>(T v);
		class GenDelegateDemo{
		public static void Main()
		{ GenericDelegate<int> intDel = Sum;
		Console.WriteLine(intDel(3));
		GenericDelegate<string> strDel = Reflect;
		Console.WriteLine(strDel("Hello"));
		}
		}	

	All the events(of Button) are declared in the class(Button) prefixed with "event" keyword and has type as "EventHandler"	public event EventHandler DoubleClick;	

Declarig Events in class		// Declare a delegate for an event.
		delegate void MyEventHandler();
		// Declare an event class.
		class MyEvent
		{
		public event MyEventHandler MyClickEvent;
		// This is called to fire the event.
		public void InvokeClickEvent()
		{ if(MyClickEvent != null)
		MyClickEvent();
		}
		}	

Difference between delegate and event	Delegate is Type safe function pouinter where Event is type of delegate.	For e.g. EventHandler is delegate in .net framework where click event is type of EventHandler	

Concepts	Discription	Example	
	There can be situaltion, We create variable of type delegate but don't use event keyword. In that case we can over write the callbacks.		
Assambly	1.Private assembly 2.Shared Assembly	GAC .. ??	
Reflection	Reflection is ability to find information about types contained in an assembly at run time.		
Attribute	•Attributes concept in .Net is a way to mark or store meta data about the code in assembly. • Often it is an instruction meant for the runtime. • The Runtime can change its behavior or course of action based on the attribute present.		
	Difference between “String” and “string”	they are the same thing and string is just an alias to String.	
C# 4.0 New Features			

1.Optional Paramenters		public void Process(string data, bool ignoreWS = false, ArrayList moreData = null)
		{
		   // Actual work done here
		}
		Process("foo"); // valid
		Process("foo", true); // valid
		Process("foo", false, myArrayList); // valid
		Process("foo", myArrayList); // Invalid! See next section
		
		Process("foo", moreData: myArrayList); // valid, ignoreWS omitted
			http://www.codeproject.com/Articles/37795/C-s-New-Features-Explained

2.Dynamic support	is used to tell the compiler that a variable's type can change or that it is not known until runtime.		
3.Co and Contra Variance	covariance and contravariance enable implicit reference conversion for array types, delegate types, and generic type arguments.Covariance preserves assignment compatibility and contravariance reverses it.		http://blogs.msdn.com/b/csharpfaq/archive/2010/02/16/covariance-and-contravariance-faq.aspx
4.COM Interop			
C# 4.5 New Features			
async and await	It will limit the functionaly of asyncronous		http://www.codeproject.com/Articles/599756/Five-Great-NET-Framework-Features
2.Zip facility (Zip compression)	Need to explore two new namespace 1.System.IO.Compression.FileSystem 2.System.IO.Compression to know what are the functionality privided by this feature		

3.Regex Timeout	Allows to set time,if reg ex match doesn’t match within that time timout excception will occour.	try
		{
		  var regEx = new Regex(@”^(\d+)+$”, RegexOptions.Singleline, TimeSpan.FromSeconds(2));
		  var match = regEx.Match (“123453109839109283090492309480329489812093809x”);
		}
		catch (RegexMatchTimeoutException ex)
		{
		  Console.WriteLine(“Regex Timeout”);
		}	

Diamond Problem			
Type vs Member Type :	classes, structs, enums, interfaces, delegates are called as types and fields, properties, constructors, methods etc., that normally reside in a type are called as type members.		
	NOTE : Type members can have all the access modifiers, where as types can have only 2 (internal, public) of the 5 access modifiers		
reason for override :	TO convert the complex type object into string is not possible with the help of toString method. We need to override the toString method and provide the default implementation.We can achieve the same thing by Convert.toString()		
Difference between Convert.ToString() and ToString() method :	If we call toString on Null object,Tostring will throw null reference exception where Convert.Tostring() doesnt.		
Equals :	Equal and "==" work the same for value type and correctly.where for reference type		
Partial classes :	Partial classes allow us to split a class into 2 or more files. All these parts are then combined into a single class, when the application is compiled. The partial keyword can also be used to split a struct or an interface over two or more files.		
Advantages of partial classes	1. The main advantage is that, visual studio uses partial classes to separate, automatically generated system code from the developer's code. For example, when you add a webform, two .CS files are generated a) WebForm1.aspx.cs - Contains the developer code b) WebForm1.aspx.designer.cs - Contains the system generated code. For example, declarations for the controls that you drag and drop on the webform. 2. When working on large projects, spreading a class over separate files allows multiple programmers to work on it simultaneously.		
Rules for Creating partial classes in c#	1.All the parts spread across different files, must use the partial keyword. Otherwise a compiler error is raised. Missing partial modifier. Another partial declaration of this type exists 2. All the parts spread across different files, must have the same access modifiers. Otherwise a compiler error is raised. Partial declarations have conflicting accessibility modifiers 3. If any of the parts are declared abstract, then the entire type is considered abstract.		
	4. If any of the parts are declared sealed, then the entire type is considered sealed. 5. If any of the parts inherit a class, then the entire type inherits that class. 6. C# does not support multiple class inheritance. Different parts of the partial class, must not specify different base classes. The following code will raise a compiler error stating - Partial declarations must not specify different base classes. 7.Different parts of the partial class can specify different base interfaces, and the final type implements all of the interfaces listed by all of the partial declarations. 8.Any members that are declared in a partial definition are available to all of the other parts of the partial class		
Partial Method :	A partial class or a struct can contain partial methods. A partial method is created using the partial keyword.	Rules http://csharp-video-tutorials.blogspot.in/2012/11/partial-methods-in-c-part-63.html	
Lambda :	A lambda expression is an anonymous function that you can use to create delegates or expression tree types. By using lambda expressions, you can write local functions that can be passed as arguments or returned as the value of function calls		

anonymous methods :	There is one case in which an anonymous method provides functionality not found in lambda expressions. Anonymous methods enable you to omit the parameter list.	delegate() {
		            System.Console.Write("Hello, ");
		            System.Console.WriteLine("World!");
		        }	

use of delegate.	1.If you don’t want to pass your interface or abstract class dependence to internal class or layers. 2.If the code doesn't need access to any other attributes or method of the class from which logic needs to be processed. 3.Event driven implementation needs to be done.		
Different Flavors of Delegate :	Func Action Predicate Converter Comparison		

Func	Func is logically similar to base delegate implementation. The difference is in the way we declare. At the time of declaration, we need to provide the signature parameter & its return type	Func<string, int, int> tempFuncPointer;
		Func<string, int, int> tempFuncPointer = tempObj. FirstTestFunction;
		 int value = tempFuncPointer("hello", 3);
		 Console.ReadKey();	

Action<TParameter>	Action is used when we do not have any return type from method. Method with void signature is being used with Action delegate.	Action<string, int> tempActionPointer;
		Action<string, int> tempActionPointer = tempObj. ThirdTestFunction;
		tempActionPointer("hello", 4);
		Console.ReadKey();	

3.Predicate<in T> :	Predicate is a function pointer for method which returns boolean value. They are commonly used for iterating a collection or to verify if the value does already exist. Declaration for the same looks like this:	Predicate<Employee> tempPredicatePointer;	
Converter<TInput, TOutput> :	Convertor delegate is used when you need to migrate / convert one collection into another by using some algorithm. Object A gets converted into Object B.	Converter<Employee, XEmployee> tempConvertorPointer = new Converter<Employee, XEmployee>(tempObj. FifthTestFunction);	
5.Comparison<T>	Comparison delegate is used to sort or order the data inside a collection. It takes two parameters as generic input type and return type should always be int. This is how we can declare Comparison delegate.	Comparison<string> tempComparison = new Comparison<string> (tempObj.SixTestFunction);	http://www.codeproject.com/Articles/741064/Delegates-its-Modern-Flavors-Func-Action-Predicate
	<Explore more. Use Titan Project to explore>		

Concepts	Discription	Example	

Immutable vs Mutable	Mutable and immutable are English words that mean "can change" and "cannot change" respectively.
	that means the mutable types are those whose data members can be changed after the instance is created but Immutable types are those whose data members can not be changed after the instance is created.
	When we change the value of mutable objects, value is changed in same memory. But in immutable type, the new memory is created and the modified value is stored in new memory.
	String is immutable and StringBuilder is mutable
		Example of immutable class:
		class MyClass
		{
		private readonly string myStr;
		public MyClass(string str)
		{
		myStr = str;
		}
		public string GetStr
		{
		get { return myStr; }
		}
		}
		1. Parameterized constructor
		2. only get accessor
			In detail: https://www.c-sharpcorner.com/article/mutable-and-immutable-class-in-c-sharp/

Dispose and Idisposable	You implement a Dispose method to release unmanaged resources used by your application. The .NET garbage collector does not allocate or release unmanaged memory. Refer Garbage collection at below for more detail	https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/implementing-dispose	
	The pattern for disposing an object, referred to as a dispose pattern, imposes order on the lifetime of an object. The dispose pattern is used only for objects that access unmanaged resources, such as file and pipe handles, registry handles, wait handles, or pointers to blocks of unmanaged memory. This is because the garbage collector is very efficient at reclaiming unused managed objects, but it is unable to reclaim unmanaged objects.	https://msdn.microsoft.com/en-us/library/system.idisposable.dispose%28v=vs.110%29.aspx	
	Dispose	Finalize	
Dispose vs Finaliize	It is used to free unmanaged resources like files, database connections etc. at any time.	It can be used to free unmanaged resources (when you implement it) like files, database connections etc. held by an object before that object is destroyed.	
	Explicitly, it is called by user code and the class which is implementing dispose method, must has to implement IDisposable interface.	Internally, it is called by Garbage Collector and cannot be called by user code.	
	It belongs to IDisposable interface.	It belongs to Object class.	
	It's implemented by implementing IDisposable interface Dispose() method.	It's implemented with the help of destructor in C++ & C#.	
	There is no performance costs associated with Dispose method.	There is performance costs associated with Finalize method since it doesn't clean the memory immediately and called by GC automatically.	
WPF vs Silver Light	WPF is a thick Windows client platform that has access to the full .Net Framework. Silverlight is a browser-based technology that has access to a subset of the .Net Framework (called the CoreCLR). So, you'll notice differences using seemingly every day methods and objects within the framework. For instance, the Split() method on the String class has 3 overrides in Silverlight, but 6 in the .Net Framework. You'll see differences like this a lot. Within WPF, all visually rendering elements derive from the Visual base class. Within Silverlight, they do not; instead, they derive from Control. Both technologies, however, eventual derive from the DependencyObject class up the hierarchy. WPF, currently, ships or has available more user controls than Silverlight; though this difference is being mitigated through the Silverlight Toolkit and the upcoming release of Silverlight 3.		
	WPF supports 3 types of routed events (direct, bubbling, and tunneling). Silverlight supports direct and bubbling only. There's quite a few data-binding differences that will be somewhat mitigated with the next version of Silverlight. Currently, Silverlight doesn't support the binding mode, OneWayToSource, or Explict UpdateSourceTriggers. In addition, Silverlight defaults to OneWay databinding if none is set, while WPF uses the default mode specified by the dependency property. Silveright doesn't support MultiBinding.		
	Silverlight supports the XmlDataProvider but not the ObjectDataProvider. WPF supports both. Silverlight can only make asynchronous network calls. WPF has access to the full .Net networking stack and can make any type of call. Also, currently, Silverlight supports SOAP, but can not handle SOAP fault exceptions natively (this may change in Silverlight 3). There are huge differences in Cryptography (Silverlight has 20 classes in the namespace, while WPF has access to 107). Basically, Silverlight supports only 4 hashing algorithms and the AES encryption protocol. Silverlight doesn't yet support: Commanding, Validation, Printing, XPS Documents, Speech, 3D, Freezable objects, or InterOp with the Windows Desktop; all of which are available in WPF. Silverlight supports browser interop, more media streaming options including timeline markers, and Deep Zoom. WPF doesn't support these features yet.		
strong naming assembly	1. Strong naming your assembly allows you to include your assembly into the Global Assembly Cache (GAC). Thus it allows you to share it among multiple applications. 2. Strong naming guarantees a unique name for that assembly. Thus no one else can use the same assembly name. 3. Strong name protect the version lineage of an assembly. A strong name can ensure that no one is able to produce a subsequent version of your assembly. Application users are ensured that a version of the assembly they are loading come from the same publisher that created the version the application was built with. 4. Strong named assemblies are signed with a digital signature. This protects the assembly from modification. Any tampering causes the verification process that occurs at assembly load time to fail. An exception is generated and the assembly is not loaded.		
Garbage collection in detail	There are three generations of objects on the heap: Generation 0. This is the youngest generation and contains short-lived objects. Generation 1. This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects. Generation 2. This generation contains long-lived objects.		https://www.codeproject.com/Articles/1095402/Garbage-Collection-and-Csharp
	Survival and promotions Objects that are not reclaimed in a garbage collection are known as survivors, and are promoted to the next generation. Objects that survive a generation 0 garbage collection are promoted to generation 1; objects that survive a generation 1 garbage collection are promoted to generation 2; and objects that survive a generation 2 garbage collection remain in generation 2.		
SOLID			
The Single Responsibility Principle	The Single Responsibility Principle (SRP) states that objects should have a single responsibility and all of their behaviors should focus on that one responsibility.		
The Open/Closed Principle			
Liskov Substitution Principle	The Liskov Substitution Principle (LSP) states that objects should be easily replaceable by instances of their subtypes without influencing the behavior and rules of the objects.		
The Interface Segregation Principle	The Interface Segregation Principle (ISP) encourages the use—and at the same time, limits the size—of interfaces throughout an application. In other words, instead of one superclass interface that contains all the behavior for an object, there should exist multiple, smaller, more specific interfaces.		
The Dependency Inversion Principle	The Dependency Inversion Principle (DIP) says that components that depend on each other should interact via an abstraction and not directly with a concrete implementation.		

Concepts	Discription	Example	
Stronged named assembly benefits	You want to enable your assemblies to be referenced by strong-named assemblies, or you want to give friend access to your assemblies from other strong-named assemblies. => Practical ?? An app needs access to different versions of the same assembly. This means you need different versions of an assembly to load side by side in the same app domain without conflict. For example, if different extensions of an API exist in assemblies that have the same simple name, strong-naming provides a unique identity for each version of the assembly. You do not want to negatively affect performance of apps using your assembly, so you want the assembly to be domain neutral. This requires strong-naming because a domain-neutral assembly must be installed in the global assembly cache. When you want to centralize servicing for your app by applying publisher policy, which means the assembly must be installed in the global assembly cache.		
What is GAC	GAC stands for Global Assembly Cache. The global assembly cache stores assemblies specifically designed to be shared by several applications on the computer. Location of GAC depends upon the version of .NET framework you are using. C:\Windows\Assembly for.NET framework 2.0 or 3.5 i.e. before 4.0 C:\WINDOWS\Microsoft.NET\assembly - For .NET framework 4.0 & above		
	With the introduction of .NET 4.0, we have 2 GAC's. One for DotNet 2.0 to 3.5 assemblies and the other for .NET 4.0 assemblies. The following are the paths for the 2 GAC's 1. C:\Windows\Assembly - For .NET 2.0 - 3.5 assemblies 2. C:\WINDOWS\Microsoft.NET\assembly - For .NET 4.0 assemblies		
How can we register/deploy dll in GAC	To install an assembly into the GAC, the assembly must be strongly named, otherwise you get an error stating - Failure adding assembly to the cache: Attempt to install an assembly without a strong name. There are 2 ways to install an assembly into GAC. 1. Simply Drag and Drop 2. Use GacUtil.exe (GAC Utility Tool)	To install an assembly using gacutil, use the following command. This command installs SampleAssembly.dll into the GAC. If you have build this project using .NET framwork 4.0 then look in C:\WINDOWS\Microsoft.NET\assembly, else look in C:\Windows\Assembly. Gacutil -i C:\SampleProject\SampleAssembly.dll To uninstall an assembly from the GAC, using GAC utility, use the following command. Gacutil -u MyClassLibrary If there are multiple versions of MyClassLibrary assembly, in the GAC, then all these versions will be removed by the above command. If you want to remove only one of the assemblies then specify the full name as shown below. gacutil -u ClassLibrary,Version=1.0.0.0, PublicKeyToken=eeaabf36d7783129	
Assembly manifest	An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE (Portable Executable) file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE (Portable Executable) file that contains only assembly manifest information. The following table shows the information contained in the assembly manifest. The first four items the assembly name, version number, culture, and strong name information make up the assembly's identity. Assembly name: A text string specifying the assembly's name.Version number: A major and minor version number, and a revision and build number. The common language runtime uses these numbers to enforce version policy.Culture: Information on the culture or language the assembly supports. This information should be used only to designate an assembly as a satellite assembly containing culture- or language-specific information. (An assembly with culture information is automatically assumed to be a satellite assembly.) Strong name information: The public key from the publisher if the assembly has been given a strong name. List of all files in the assembly:		
Benefits of strong naming assembly	Strong naming your assembly allows you to include your assembly into the Global Assembly Cache (GAC). Thus it allows you to share it among multiple applications. Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key. Strong name protect the version lineage of an assembly. A strong name can ensure that no one is able to produce a subsequent version of your assembly. Application users are ensured that a version of the assembly they are loading come from the same publisher that created the version the application was built with. Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust like that provided, for example, by a digital signature and supporting certificate		

Version information	// Version information for an assembly consists of the following four values:
	//
	// Major Version
	// Minor Version
	// Build Number
	// Revision		

Generating key to created strongly named assembly	sn.Exe is used to create public private key. In Visual Studio=> Project => Properties => signing tab help to create the key and create strongly named assembly	https://docs.microsoft.com/en-us/dotnet/framework/app-domains/how-to-create-a-public-private-key-pair	
How to sign the assembly with strong name			
What if two different assembly with same namespace, classes, version and key tries to register to GAC on same machine ?			
How to access GAC registered custom dll ?			
Do I required public key to access strongly named assembly?			
What is friend access to dll ?			
What is DLL Hell	Many of the problems that lead developers to use the term "DLL hell" involve instances when an alteration to a DLL file by a program negatively affects the function of other programs that need to use the same DLL file. Problems with registries, incompatibility and the incorrect updating of DLL files are all part of the general challenge of ordering the use of DLL files across many different applications. In more current versions of Windows, some of the problems that contribute to DLL hell have been addressed and solved to some extent. Changes include a .NET framework, which uses metadata to describe program components. This system helps with versioning and deployment to alleviate some of the problems that arise due to cross-language DLL use or situations where applications have to share a DLL file		

Multi Threading	multithreading supports parallel execution of code. A thread is an independent execution path, able to run simultaneously with other threads.
	Once started, a thread?s IsAlive property returns true, until the point where the thread ends.
	Once ended, a thread cannot restart.
	The CLR assigns each thread its own memory stack so that local variables are kept separate.
	A separate copy of the cycles variable (i in this example) is created on each thread's memory stack
		class ThreadTest
		{
		static void Main()
		{
		Thread t = new Thread (WriteY); // Kick off a new thread
		t.Start(); // running WriteY()
		// Simultaneously, do something on the main thread.
		for (int i = 0; i < 1000; i++) Console.Write ("x");
		}
		static void WriteY()
		{
		for (int i = 0; i < 1000; i++) Console.Write ("y");
		}
		}
			http://www.albahari.com/threading/part5.aspx#_The_Parallel_Class

Join	You can wait for another thread to end by calling its Join method.		

Sleep	Thread.Sleep pauses the current thread for a specified period	Thread.Sleep (TimeSpan.FromHours (1)); // sleep for 1 hour
		Thread.Sleep (500);	

How Threading Works	Multithreading is managed internally by a thread scheduler, a function the CLR typically delegates to the operating system. A thread scheduler ensures all active threads are allocated appropriate execution time, and that threads that are waiting or blocked (for instance, on an exclusive lock or on user input) do not consume CPU time.		
	On a single-processor computer, a thread scheduler performs time-slicing ? rapidly switching execution between each of the active threads. Under Windows, a time-slice is typically in the tens-of-milliseconds region ? much larger than the CPU overhead in actually switching context between one thread and another (which is typically in the few-microseconds region).		

Concepts	Discription	Example	
	On a multi-processor computer, multithreading is implemented with a mixture of time-slicing and genuine concurrency, where different threads run code simultaneously on different CPUs. It?s almost certain there will still be some time-slicing, because of the operating system?s need to service its own threads ? as well as those of other applications.		
	A thread is said to be preempted when its execution is interrupted due to an external factor such as time-slicing. In most situations, a thread has no control over when and where it?s preempted.		
Threads vs Processes	A thread is analogous to the operating system process in which your application runs. Just as processes run in parallel on a computer, threads run in parallel within a single process. Processes are fully isolated from each other; threads have just a limited degree of isolation. In particular, threads share (heap) memory with other threads running in the same application. This, in part, is why threading is useful: one thread can fetch data in the background, for instance, while another thread can display the data as it arrives		
Foreground and Background Threads	By default, threads you create explicitly are foreground threads. Foreground threads keep the application alive for as long as any one of them is running, whereas background threads do not. Once all foreground threads finish, the application ends, and any background threads still running abruptly terminate.		
	You can query or change a thread?s background status using its IsBackground property		
Thread Priority	A thread?s Priority property determines how much execution time it gets relative to other active threads in the operating system, on the following scale:	enum ThreadPriority { Lowest, BelowNormal, Normal, AboveNormal, Highest }	
Thread Pool			
Why Thread Pool	Whenever you start a thread, a few hundred microseconds are spent organizing such things as a fresh private local variable stack. Each thread also consumes (by default) around 1 MB of memory. The thread pool cuts these overheads by sharing and recycling threads, allowing multithreading to be applied at a very granular level without a performance penalty. This is useful when leveraging multicore processors to execute computationally intensive code in parallel in ?divide-and-conquer? style.	Reduce the overhead of memory alloction and thread initialition and improves the performance	
There are a number of ways to enter the thread pool:	Via the Task Parallel Library (from Framework 4.0) By calling ThreadPool.QueueUserWorkItem Via asynchronous delegates Via BackgroundWorker		
There are a few things to be wary of when using pooled threads:	You cannot set the Name of a pooled thread, making debugging more difficult (although you can attach a description when debugging in Visual Studio?s Threads window). Pooled threads are always background threads (this is usually not a problem). Blocking a pooled thread may trigger additional latency in the early life of an application unless you call ThreadPool.SetMinThreads (see Optimizing the Thread Pool). You are free to change the priority of a pooled thread ? it will be restored to normal when released back to the pool.		
	You can query if you?re currently executing on a pooled thread via the property Thread.CurrentThread.IsThreadPoolThread		

Entering the Thread Pool via TPL	You can enter the thread pool easily using the Task classes in the Task Parallel Library. The Task classes were introduced in Framework 4.0: if you?re familiar with the older constructs, consider the nongeneric Task class a replacement for ThreadPool. QueueUserWorkItem, and the generic Task<TResult> a replacement for asynchronous delegates. The newer constructs are faster, more convenient, and more flexible than the old.
	
	Task.Factory.StartNew returns a Task object, which you can then use to monitor the task ? for instance, you can wait for it to complete by calling its Wait method.
		static void Main() // The Task class is in System.Threading.Tasks
		{
		Task.Factory.StartNew (Go);
		}
		
		static void Go()
		{
		Console.WriteLine ("Hello from the thread pool!");
		}	

ThreadPool.QueueUserWorkItem	To use QueueUserWorkItem, simply call this method with a delegate that you want to run on a pooled thread.
	Our target method, Go, must accept a single object argument (to satisfy the WaitCallback delegate). This provides a convenient way of passing data to the method, just like with ParameterizedThreadStart. Unlike with Task, QueueUserWorkItem doesn't return an object to help you subsequently manage execution. Also, you must explicitly deal with exceptions in the target code ? unhandled exceptions will take down the program.
		static void Main()
		{
		ThreadPool.QueueUserWorkItem (Go);
		ThreadPool.QueueUserWorkItem (Go, 123);
		Console.ReadLine();
		}
		
		static void Go (object data) // data will be null with the first call.
		{
		Console.WriteLine ("Hello from the thread pool! " + data);
		}	

Asynchronous delegates	ThreadPool.QueueUserWorkItem doesn?t provide an easy mechanism for getting return values back from a thread after it has finished executing. Asynchronous delegate invocations (asynchronous delegates for short) solve this, allowing any number of typed arguments to be passed in both directions. Furthermore, unhandled exceptions on asynchronous delegates are conveniently rethrown on the original thread (or more accurately, the thread that calls EndInvoke), and so they don?t need explicit handling.	static void Main()
		{
		Func<string, int> method = Work;
		IAsyncResult cookie = method.BeginInvoke ("test", null, null);
		//
		// ... here's where we can do other work in parallel...
		//
		int result = method.EndInvoke (cookie);
		Console.WriteLine ("String length is: " + result);
		}
		
		static int Work (string s) { return s.Length; }	

	Here?s how you start a worker task via an asynchronous delegate: Instantiate a delegate targeting the method you want to run in parallel (typically one of the predefined Func delegates). Call BeginInvoke on the delegate, saving its IAsyncResult return value. BeginInvoke returns immediately to the caller. You can then perform other activities while the pooled thread is working. When you need the results, call EndInvoke on the delegate, passing in the saved IAsyncResult object.		
	EndInvoke does three things. First, it waits for the asynchronous delegate to finish executing, if it hasn?t already. Second, it receives the return value (as well as any ref or out parameters). Third, it throws any unhandled worker exception back to the calling thread.		
synchronization	coordinating the actions of threads for a predictable outcome. Synchronization is particularly important when threads access the same data; it?s surprisingly easy to run aground in this area.		
Synchronization constructs can be divided into four categories	Simple blocking methods These wait for another thread to finish or for a period of time to elapse. Sleep, Join, and Task.Wait are simple blocking methods. Locking constructs These limit the number of threads that can perform some activity or execute a section of code at a time. Exclusive locking constructs are most common ? these allow just one thread in at a time, and allow competing threads to access common data without interfering with each other. The standard exclusive locking constructs are lock (Monitor.Enter/Monitor.Exit), Mutex, and SpinLock. The nonexclusive locking constructs are Semaphore, SemaphoreSlim, and the reader/writer locks.		
	Signaling constructs These allow a thread to pause until receiving a notification from another, avoiding the need for inefficient polling. There are two commonly used signaling devices: event wait handles and Monitor?s Wait/Pulse methods. Framework 4.0 introduces the CountdownEvent and Barrier classes. Nonblocking synchronization constructs These protect access to a common field by calling upon processor primitives. The CLR and C# provide the following nonblocking constructs: Thread.MemoryBarrier, Thread.VolatileRead, Thread.VolatileWrite, the volatile keyword, and the Interlocked class.		
Locking	Exclusive locking is used to ensure that only one thread can enter particular sections of code at a time.		

	Only one thread can lock the synchronizing object (in this case, _locker) at a time, and any contending threads are blocked until the lock is released. If more than one thread contends the lock, they are queued on a ?ready queue? and granted the lock on a first-come, first-served basis	static readonly object _locker = new object();
		static void Go()
		{
		lock (_locker)
		{
		if (_val2 != 0) Console.WriteLine (_val1 / _val2);
		_val2 = 0;
		}
		}	

Monitor.Enter and Monitor.Exit	C#?s lock statement is in fact a syntactic shortcut for a call to the methods Monitor. Enter and Monitor.Exit, with a try/finally block. Here?s (a simplified version of) what?s actually happening within the Go method of the preceding example:	Monitor.Enter (_locker);
		try
		{
		if (_val2 != 0) Console.WriteLine (_val1 / _val2);
		_val2 = 0;
		}
		finally { Monitor.Exit (_locker); }	

Concepts	Discription	Example	
Synchronization object	Any reference object can be used as synchronized object.		
Mutex	A Mutex is like a C# lock, but it can work across multiple processes. In other words, Mutex can be computer-wide as well as application-wide.		
	With a Mutex class, you call the WaitOne method to lock and ReleaseMutex to unlock. Closing or disposing a Mutex automatically releases it. Just as with the lock statement, a Mutex can be released only from the same thread that obtained it.		
Named Mutex			

Semaphore	A semaphore is like a nightclub: it has a certain capacity, enforced by a bouncer. Once it?s full, no more people can enter, and a queue builds up outside. Then, for each person that leaves, one person enters from the head of the queue. The constructor requires a minimum of two arguments: the number of places currently available in the nightclub and the club?s total capacity.
	
	A semaphore with a capacity of one is similar to a Mutex or lock, except that the semaphore has no ?owner? ? it?s thread-agnostic. Any thread can call Release on a Semaphore, whereas with Mutex and lock, only the thread that obtained the lock can release it.
	Semaphores can be useful in limiting concurrency ? preventing too many threads from executing a particular piece of code at once.
		class TheClub // No door lists!
		{
		static SemaphoreSlim _sem = new SemaphoreSlim (3); // Capacity of 3
		
		static void Main()
		{
		for (int i = 1; i <= 5; i++) new Thread (Enter).Start (i);
		}
		
		static void Enter (object id)
		{
		Console.WriteLine (id + " wants to enter");
		_sem.Wait();
		Console.WriteLine (id + " is in!"); // Only three threads
		Thread.Sleep (1000 * (int) id); // can be here at
		Console.WriteLine (id + " is leaving"); // a time.
		_sem.Release();
		}
		}	

		1 wants to enter 1 is in! 2 wants to enter 2 is in! 3 wants to enter 3 is in! 4 wants to enter 5 wants to enter 1 is leaving 4 is in! 2 is leaving 5 is in!	
Rich Client Applications and Thread Affinity	Both the Windows Presentation Foundation (WPF) and Windows Forms libraries follow models based on thread affinity. Although each has a separate implementation, they are both very similar in how they function. The objects that make up a rich client are based primarily on DependencyObject in the case of WPF, or Control in the case of Windows Forms. These objects have thread affinity, which means that only the thread that instantiates them can subsequently access their members. Violating this causes either unpredictable behavior, or an exception to be thrown. On the positive side, this means you don?t need to lock around accessing a UI object. On the negative side, if you want to call a member on object X created on another thread Y, you must marshal the request to thread Y. You can do this explicitly as follows: In WPF, call Invoke or BeginInvoke on the element?s Dispatcher object. In Windows Forms, call Invoke or BeginInvoke on the control. Invoke and BeginInvoke both accept a delegate, which references the method on the target control that you want to run. Invoke works synchronously: the caller blocks until the marshal is complete. BeginInvoke works asynchronously: the caller returns immediately and the marshaled request is queued up (using the same message queue that handles keyboard, mouse, and timer events)		

Synchronization Contexts	An alternative to locking manually is to lock declaratively. By deriving from ContextBoundObject and applying the Synchronization attribute, you instruct the CLR to apply locking automatically.
	The CLR ensures that only one thread can execute code in safeInstance at a time. It does this by creating a single synchronizing object ? and locking it around every call to each of safeInstance's methods or properties. The scope of the lock ? in this case, the safeInstance object ? is called a synchronization context.
	
	So, how does this work? A clue is in the Synchronization attribute's namespace: System.Runtime.Remoting.Contexts. A ContextBoundObject can be thought of as a ?remote? object, meaning all method calls are intercepted. To make this interception possible, when we instantiate AutoLock, the CLR actually returns a proxy ? an object with the same methods and properties of an AutoLock object, which acts as an intermediary. It's via this intermediary that the automatic locking takes place. Overall, the interception adds around a microsecond to each method call.
		using System;
		using System.Threading;
		using System.Runtime.Remoting.Contexts;
		
		[Synchronization]
		public class AutoLock : ContextBoundObject
		{
		public void Demo()
		{
		Console.Write ("Start...");
		Thread.Sleep (1000); // We can't be preempted here
		Console.WriteLine ("end"); // thanks to automatic locking!
		}
		}
		
		public class Test
		{
		public static void Main()
		{
		AutoLock safeInstance = new AutoLock();
		new Thread (safeInstance.Demo).Start(); // Call the Demo
		new Thread (safeInstance.Demo).Start(); // method 3 times
		safeInstance.Demo(); // concurrently.
		}
		}	

		//Output Start... end Start... end Start... En	
Thread (Thread Spawning) Vs Thread Pool	The problem with creating your own threads Creating and destroying threads has a high CPU usage, so when you need to perform lots of small, simple tasks concurrently the overhead of creating your own threads can take up a significant portion of the CPU cycles and severely affect the final response time. This is especially true in stress conditions where executing multiple threads can push CPU to 100% and most of the time would be wasted in context switching		https://theburningmonk.com/2010/03/threading-using-the-threadpool-vs-creating-your-own-threads/
	Using the Thread Pool This is where the .Net Thread Pool comes in, where a number of threads are created ahead of time and kept around to pick up any work items you give them to do, without the overhead associated with creating your own threads.		
When not to use the Thread Pool	You require a foreground thread, all the thread pool threads are background threads You require a thread to have a particular priority. You have tasks that cause the thread to block for long periods of time. The thread pool has a maximum number of threads, so a large number of blocked thread pool threads might prevent tasks from starting. You need to place threads into a single-threaded apartment. All ThreadPool threads are in the multithreaded apartment. You need to have a stable identity associated with the thread, or to dedicate a thread to a task. There is no easy way to detect that a threadpool completed, no Thread.Join() There is no easy way to marshal exceptions from a threadpool thread You cannot abort a threadpool thread		
Monitor Vs Lock	<a good interview question> :)		https://www.interviewsansar.com/2016/06/23/monitor-vs-lock-in-csharp-why-use-monitor-instead-of-lock/
Lock Vs Mutex	A lock allows only one thread to enter the part that's locked and the lock is not shared with any other processes. A mutex is the same as a lock but it can be system wide (shared by multiple processes). A semaphore does the same as a mutex but allows x number of threads to enter, this can be used for example to limit the number of cpu, io or ram intensive tasks running at the same time.		
Mutex Vs Semaphore			https://stackoverflow.com/questions/34519/what-is-a-semaphore/40238#40238

Concepts	Discription	Example	
BackroundWorker	BackgroundWorker is a helper class in the System.ComponentModel namespace for managing a worker thread. It can be considered a general-purpose implementation of the EAP(Event-Based Asynchronous Pattern), and provides the following features: A cooperative cancellation model The ability to safely update WPF or Windows Forms controls when the worker completes (RunWorkerCompletedEventHandler) Forwarding of exceptions to the completion event A protocol for reporting progress (ProgressChangedEventHandler) An implementation of IComponent allowing it to be sited in Visual Studio?s designer (public class BackgroundWorker : Component)		
BackroundWorker Vs Thread			

Safe Cancellation using CancellationToken	cooperative Cancellation	var cancelSource = new CancellationTokenSource();
		new Thread (() => Work (cancelSource.Token)).Start();
		
		void Work (CancellationToken cancelToken)
		{
		cancelToken.ThrowIfCancellationRequested();
		...
		}	

Lazy Initialization	Like lazy initiallize singleton object		

Lazy Initialization using Lazy<T>		Lazy<Expensive> _expensive = new Lazy<Expensive>
		(() => new Expensive(), true);
		
		public Expensive Expensive { get { return _expensive.Value; } }	

LazyInitializer	LazyInitializer is a static class that works exactly like Lazy<T> except:
	
	Its functionality is exposed through a static method that operates directly on a field in your own type. This avoids a level of indirection, improving performance in cases where you need extreme optimization.
	It offers another mode of initialization that has multiple threads race to initialize.
		Expensive _expensive;
		public Expensive Expensive
		{
		get // Implement double-checked locking
		{
		LazyInitializer.EnsureInitialized (ref _expensive,
		() => new Expensive());
		return _expensive;
		}
		}	

Thread-Local Storage	Thread-local storage (TLS) is a computer programming method that uses static or global memory local to a thread. All threads of a process share the virtual address space of the process. The local variables of a function are unique to each thread that runs the function. However, the static and global variables are shared by all threads in the process. With thread local storage (TLS), you can provide unique data for each thread that the process can access using a global index. One thread allocates the index, which can be used by the other threads to retrieve the unique data associated with the index.	<Real case scenario where Thread local storage is required>	
	There are three ways to implement thread-local storage		
[ThreadStatic]	The easiest approach to thread-local storage is to mark a static field with the ThreadStatic attribute:	[ThreadStatic] static int _x	
	Unfortunately, [ThreadStatic] doesn?t work with instance fields (it simply does nothing); nor does it play well with field initializers ? they execute only once on the thread that's running when the static constructor executes.		
GetData and SetData	<Explore more if important?>		
PFX : parallel programming	Programming to leverage multicores or multiple processors is called parallel programming. This is a subset of the broader concept of multithreading.		
	There are two strategies for partitioning work among threads: data parallelism and task parallelism. When a set of tasks must be performed on many data values, we can parallelize by having each thread perform the (same) set of tasks on a subset of values. This is called data parallelism because we are partitioning the data between threads. In contrast, with task parallelism we partition the tasks; in other words, we have each thread perform a different task.		
Plinq	Parallel LINQ (PLINQ) is a parallel implementation of the LINQ pattern. The System.Linq.ParallelEnumerable class exposes almost all of PLINQ's functionality. ParallelEnumerable includes implementations of all the standard query operators that LINQ to Objects supports, although it does not attempt to parallelize each one.		
PLINQ-specific methods are listed in the following table.	AsParallel : The entry point for PLINQ. Specifies that the rest of the query should be parallelized, if it is possible. AsSequential : Specifies that the rest of the query should be run sequentially, as a non-parallel LINQ query. AsOrdered : Specifies that PLINQ should preserve the ordering of the source sequence for the rest of the query, or until the ordering is changed, for example by the use of an orderby (Order By in Visual Basic) clause.		https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-to-plinq
	AsUnordered : Specifies that PLINQ for the rest of the query is not required to preserve the ordering of the source sequence. WithCancellation : Specifies that PLINQ should periodically monitor the state of the provided cancellation token and cancel execution if it is requested. WithDegreeOfParallelism : Specifies the maximum number of processors that PLINQ should use to parallelize the query.		
	WithMergeOptions: Provides a hint about how PLINQ should, if it is possible, merge parallel results back into just one sequence on the consuming thread. WithExecutionMode: Specifies whether PLINQ should parallelize the query even when the default behavior would be to run it sequentially. ForAll: A multithreaded enumeration method that, unlike iterating over the results of the query, enables results to be processed in parallel without first merging back to the consumer thread. Aggregate: overload An overload that is unique to PLINQ and enables intermediate aggregation over thread-local partitions, plus a final aggregation function to combine the results of all partitions.		
PLINQ Limitations	The following query operators prevent a query from being parallelized, unless the source elements are in their original indexing position: Take, TakeWhile, Skip, and SkipWhile The indexed versions of Select, SelectMany, and ElementAt		
	The following query operators are parallelizable, but use an expensive partitioning strategy that can sometimes be slower than sequential processing: Join, GroupBy, GroupJoin, Distinct, Union, Intersect, and Except		
PLINQ has three partitioning strategies for assigning input elements to threads:	Strategy	Element allocation	Relative performance
	Chunk partitioning	Dynamic	Average
	Range partitioning	Static	Poor to excellent
	Hash partitioning	Static	Poor
The Parallel Class	PFX provides a basic form of structured parallelism via three static methods in the Parallel class:		
	Parallel.Invoke Executes an array of delegates in parallel Parallel.For Performs the parallel equivalent of a C# for loop Parallel.ForEach Performs the parallel equivalent of a C# foreach loop		
Task Parallelism	Task parallelism is the lowest-level approach to parallelization with PFX. The classes for working at this level are defined in the System.Threading.Tasks namespace and comprise the following:		
	Class => Purpose Task : For managing a unit for work Task<TResult> : For managing a unit for work with a return value TaskFactory : For creating tasks TaskFactory<TResult> : For creating tasks and continuations with the same return type TaskScheduler : For managing the scheduling of tasks TaskCompletionSource : For manually controlling a task?s workflow	a task is a lightweight object for managing a parallelizable unit of work. A task avoids the overhead of starting a dedicated thread by using the CLR?s thread pool: this is the same thread pool used by ThreadPool.QueueUserWorkItem, Tasks can be used whenever you want to execute something in parallel. However, they?re tuned for leveraging multicores: in fact, the Parallel class and PLINQ are internally built on the task parallelism constructs.	
some powerful features	Tune a task?s scheduling Establish a parent/child relationship when one task is started from another Implement cooperative cancellation Wait on a set of tasks ? without a signaling construct Attach ?continuation? task(s) Schedule a continuation based on multiple antecedent tasks Propagate exceptions to parents, continuations, and task consumers		
	The static TaskScheduler.UnobservedTaskException event provides a final last resort for dealing with unhandled task exceptions. By handling this event, you can intercept task exceptions that would otherwise end the application ? and provide your own logic for dealing with them.		
TaskCreationOptions			

Concepts	Discription	Example	
Waiting on Tasks	Calling its Wait method (optionally with a timeout) Accessing its Result property (in the case of Task<TResult>)		
	You can also wait on multiple tasks at once ? via the static methods Task.WaitAll (waits for all the specified tasks to finish) and Task.WaitAny (waits for just one task to finish)		

Continuations	Sometimes it?s useful to start a task right after another one completes (or fails). The ContinueWith method on the Task class does exactly this:	Task task1 = Task.Factory.StartNew (() => Console.Write ("antecedant.."));
		Task task2 = task1.ContinueWith (ant => Console.Write (".. continuation"));	

	Another (subtler) difference is that by default, antecedent and continuation tasks may execute on different threads. You can force them to execute on the same thread by specifying TaskContinuationOptions.ExecuteSynchronously when calling ContinueWith: this can improve performance in very fine-grained continuations by lessening indirection.		
Task Schedulers	A task scheduler allocates tasks to threads. All tasks are associated with a task scheduler, which is represented by the abstract TaskScheduler class. The Framework provides two concrete implementations: the default scheduler that works in tandem with the CLR thread pool, and the synchronization context scheduler. The latter is designed (primarily) to help you with the threading model of WPF and Windows Forms, which requires that UI elements and controls are accessed only from the thread that created them.		
TaskFactory	When you call Task.Factory, you?re calling a static property on Task that returns a default TaskFactory object. The purpose of a task factory is to create tasks		
Flatten and Handle of AggregateException			
Concurrent Collections	ConcurrentStack<T> => Stack<T> ConcurrentQueue<T> => Queue<T> ConcurrentBag<T> (none) BlockingCollection<T> (none) ConcurrentDictionary<TKey,TValue>		
TaskCreationOptions,TaskCreationOptions, TaskScheduler			
Expression in LINQ	We have learned that the lambda Expression can be assigned to the Func or Action type delegates to process over in-memory collections. The .NET compiler converts the lambda expression assigned to Func or Action type delegate into executable code at compile time.		
	LINQ introduced the new type called Expression that represents strongly typed lambda expression. It means lambda expression can also be assigned to Expression<TDelegate> type. The .NET compiler converts the lambda expression which is assigned to Expression<TDelegate> into an Expression tree instead of executable code. This expression tree is used by remote LINQ query providers as a data structure to build a runtime query out of it (such as LINQ-to-SQL, EntityFramework or any other LINQ query provider that implements IQueryable<T> interface).		https://www.tutorialsteacher.com/linq/linq-expression
	can convert the Func type delegate into an Expression by wrapping Func delegate with Expresson	Expression<Func<Student, bool>> isTeenAgerExpr = s => s.age > 12 && s.age < 20;	

	You can invoke the delegate wrapped by an Expression the same way as a delegate, but first you need to compile it using the Compile() method. Compile() returns delegateof Func or Action type so that you can invoke it like a delegate	Expression<Func<Student, bool>> isTeenAgerExpr = s => s.age > 12 && s.age < 20;
		
		//compile Expression using Compile method to invoke it as Delegate
		Func<Student, bool> isTeenAger = isTeenAgerExpr.Compile();
		
		//Invoke
		bool result = isTeenAger(new Student(){ StudentID = 1, StudentName = "Steve", Age = 20});	

Expression Tree	Expression tree as name suggests is nothing but expressions arranged in a tree-like data structure. Each node in an expression tree is an expression. For example, an expression tree can be used to represent mathematical formula x < y where x, < and y will be represented as an expression and arranged in the tree like structure.
	
	Expression tree is an in-memory representation of a lambda expression. It holds the actual elements of the query, not the result of the query.
	
	The expression tree makes the structure of the lambda expression transparent and explicit. You can interact with the data in the expression tree just as you can with any other data structure.
		Expression<Func<Student, bool>> isTeenAgerExpr = s => s.Age > 12 && s.Age < 20;
		
		Console.WriteLine("Expression: {0}", isTeenAgerExpr);
		
		Console.WriteLine("Expression Type: {0}", isTeenAgerExpr. NodeType);
		
		var parameters = isTeenAgerExpr.Parameters;
		
		foreach (var param in parameters)
		{
		Console.WriteLine("Parameter Name: {0}", param.Name);
		Console.WriteLine("Parameter Type: {0}", param.Type.Name);
		}
		var bodyExpr = isTeenAgerExpr.Body as BinaryExpression;
		
		Console.WriteLine("Left side of body expression: {0}", bodyExpr. Left);
		Console.WriteLine("Binary Expression Type: {0}", bodyExpr. NodeType);
		Console.WriteLine("Right side of body expression: {0}", bodyExpr. Right);
		Console.WriteLine("Return Type: {0}", isTeenAgerExpr. ReturnType);
		
		//Output
		Expression: s => ((s.Age > 12) AndAlso (s.Age < 20))
		Expression Type: Lambda
		Parameter Name: s
		Parameter Type: Student
		Left side of body expression: (s.Age > 12)
		Binary Expression Type: AndAlso
		Right side of body expression: (s.Age < 20)
		Return Type: System.Boolean	

	Executable code excutes in the same application domain to process over in-memory collection. Enumerable static classes contain extension methods for in-memory collections that implements IEnumerable<T> interface e.g. List<T>, Dictionary<T>, etc. The Extension methods in an Enumerable class accept a predicate parameter of Func type delegate. For example, the Where extension method accepts Func<TSource, bool> predicate. It then compiles it into IL (Intermediate Language) to process over in-memory collections that are in the same AppDomain.		
	Func delegate is a raw executable code, so if you debug the code, you will find that the Func delegate will be represented as opaque code. You cannot see its parameters, return type and body:		
	Func delegate is for in-memory collections because it will be processed in the same AppDomain, but what about remote LINQ query providers like LINQ-to-SQL, EntityFramework or other third party products that provides LINQ capabilities? How would they parse lambda expression that has been compiled into raw executable code to know about the parameters, return type of lambda expression and build runtime query to process further? The answer is Expression tree. Expression<TDelegate> is compiled into a data structure called an expression tree.		
	An expression tree is transparent. You can retrieve a parameter, return type and body expression information from the expression		

Concepts	Discription	Example	
Web API using System.Web.Http assembly			
Routing			
Asp.Net Web API VS Asp.Net MVC	1. Asp.Net MVC is used to create web applications that returns both views and data but Asp.Net Web API is used to create full blown HTTP services with easy and simple way that returns only data not view.		
	2. Web API helps to build REST-ful services over the .NET Framework and it also support content-negotiation(it's about deciding the best response format data that could be acceptable by the client. it could be JSON,XML, ATOM or other formatted data), self hosting which are not in MVC.		
	3. Web API also takes care of returning data in particular format like JSON, XML or any other based upon the Accept header in the request and you don't worry about that. MVC only return data in JSON format using JsonResult.		
	In Web API the request are mapped to the actions based on HTTP verbs but in MVC it is mapped to actions name.		
	Asp.Net Web API is new framework and part of the core ASP.NET framework. The model binding, filters, routing and others MVC features exist in Web API are different from MVC and exists in the new System.Web.Http assembly. In MVC, these featues exist with in System.Web.Mvc. Hence Web API can also be used with Asp.Net and as a stand alone service layer.		
	You can mix Web API and MVC controller in a single project to handle advanced AJAX requests which may return data in JSON, XML or any others format and building a full blown HTTP service. Typically, this will be called Web API self hosting.		
	When you have mixed MVC and Web API controller and you want to implement the authorization then you have to create two filters one for MVC and another for Web API since boths are different.		
	Moreover, Web API is light weight architecture and except the web application it can also be used with smart phone apps.		
	clients are required to know all of the available actions ahead of time. This means there is an implicit binding between client and server, in that the caller is dependent on a contract and a given set of actions from the service.		
ApiController	This base class was built specifically for enabling RESTful services, and you simply return the object (or, objects in a collection) of the data being requested		
1.1 REST Architecture and Its contraints			
REST	REST (REpresentational State Transfer) is an architectural style for developing web services.	Source: https://en.wikipedia.org/wiki/Representational_state_transfer	
	REST uses HTTP to communicate		
REST Architectural constraints	Six guiding constraints define a RESTful system. These constraints restrict the ways that the server can process and respond to client requests so that, by operating within these constraints, the service gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If a service violates any of the required constraints, it cannot be considered RESTful.		
1. Client–server architecture	The principle behind the client–server constraints is the separation of concerns. Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms. however, is that the separation allows the components to evolve independently,		
2. Statelessness	The client–server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client(Not a mandatory clause). The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that can be used the next time the client chooses to initiate a new state-transition	Need more Info : The representation of each application state contains links that can be used the next time the client chooses to initiate a new state-transition	
3. Cacheability	As on the World Wide Web, clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.	How to achieve this?	
4. Layered system	A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers can improve system scalability by enabling load balancing and by providing shared caches. They can also enforce security policies.		
5			
6. Uniform interface	The uniform interface constraint is fundamental to the design of any REST service. It simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are:		
Resource		The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server could send data from its database as HTML, XML or as JSON—none of which are the server's internal representation.	
6.1 Resource identification in requests	Individual resources are identified in requests, for example using URIs in Web-based REST systems.		
6.2 Resource manipulation through representations	When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.	Need to dig into	
6.3 Self-descriptive messages	Each message includes enough information to describe how to process the message. For example, which parser to invoke can be specified by a media type.	Need to dig into Message ?	
6.4 Hypermedia as the engine of application state (HATEOAS)		Need to dig into	
Request-Response Pipeline		https://exceptionnotfound.net/the-asp-net-web-api-2-http-message-lifecycle-in-43-easy-steps-2/	
Server Code		HTTP Server Code : https://github.com/ASP-NET-MVC/aspnetwebstack	
What is message handler	A message handler is a class that receives an HTTP request and returns an HTTP response. Message handlers derive from the abstract HttpMessageHandler class.		
What is delegating handler	Typically, a series of message handlers are chained together. The first handler receives an HTTP request, does some processing, and gives the request to the next handler. At some point, the response is created and goes back up the chain. This pattern is called a delegating handler.	File:HttpClientFactory	
Step 1: IIS hosting or self-hosting			

Step 2:
HttpServer:
	Http Server is type of DeligatingHandler.
	Role/Functionality:
	1. request.SetSynchronizationContext(context);
	// The first request initializes the server
	2. SynchronizationContext context = SynchronizationContext.Current;// Capture current synchronization context and add it as a parameter to the request
	request.Properties[HttpPropertyKeys.SynchronizationContextKey] = synchronizationContext;
	3. request.SetConfiguration(_configuration); // Add HttpConfiguration object as a parameter to the request
	4. // Ensure we have a principal, even if the host didn't give us one
	private static readonly IPrincipal _anonymousPrincipal = new GenericPrincipal(new GenericIdentity(String.Empty), new string[0]);
	IPrincipal originalPrincipal = Thread.CurrentPrincipal;
	
	if (originalPrincipal == null)
	{
	Thread.CurrentPrincipal = _anonymousPrincipal;
	}
		Inheritance Hierarchy
		System.Object
		System.Net.Http.HttpMessageHandler
		System.Net.Http.DelegatingHandler
		System.Web.Http.HttpServer
		System.Web.Http.SelfHost.HttpSelfHostServer	

	5. // Ensure we have a principal on the request context (if there is a request context).
	
	HttpRequestContext requestContext = request.GetRequestContext();
	if (requestContext == null)
	{
	requestContext = new RequestBackedHttpRequestContext(request);
	// if the host did not set a request context we will also set it back to the request.
	request.SetRequestContext(requestContext);
	}
	6. => Requesrt Pipeline
	7. => Exception Handling
	8. => Log Handling
		public class HttpServer : DelegatingHandler
		{
		//Dispatches an incoming HttpRequestMessage.(Overrides DelegatingHandler.SendAsync(HttpRequestMessage, CancellationToken).)
		//SendAsync(HttpRequestMessage, CancellationToken)
		}	

Step 3 & 4:
HttpRequestMessage
	HttpRequestMessage: Represents a HTTP request message.	Inheritance Hierarchy
		System.Object
		System.Net.Http.HttpRequestMessage
		public class HttpRequestMessage : IDisposable {
		public HttpContent Content { get; set; }
		public HttpRequestHeaders Headers { get; }
		public HttpMethod Method { get; set; }
		public IDictionary<string, object> Properties { get; }
		public Uri RequestUri { get; set; }
		public Version Version { get; set; }
		}	

Step 5: DelegatingHandler :	A base type for HTTP handlers that delegate the processing of HTTP response messages to another handler, called the inner handler.	Inheritance Hierarchy System.Object System.Net.Http.HttpMessageHandler System.Net.Http.DelegatingHandler System.Net.Http.MessageProcessingHandler public abstract class DelegatingHandler : HttpMessageHandler {}	
HttpConfiguration	Explore		

Step 6:
HttpRoutingDispatcher
	Role:
	1. Find the route(Route Handler) in Request
	2. if it does not find in Request, search Route Handler in config.Routes
	3. if it does not find then create the Object of HttpMessageInvoker with default Handler else create the object of HttpMessageInvoker with Route Specific.
	4. calls SendAsync on HttpMessageInvoker object
		public class HttpRoutingDispatcher : HttpMessageHandler
		{
		protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)
		{
		// Lookup route data, or if not found as a request property then we look it up in the route table
		IHttpRouteData routeData = request.GetRouteData();
		if (routeData == null)
		{
		routeData = _configuration.Routes.GetRouteData(request);
		if (routeData != null)
		{
		request.SetRouteData(routeData);
		}
		}	

		if (routeData == null || (routeData.Route != null && routeData.Route.Handler is StopRoutingHandler))
		{
		request.Properties.Add(HttpPropertyKeys.NoRouteMatched, true);
		return Task.FromResult(request.CreateErrorResponse(
		HttpStatusCode.NotFound,
		Error.Format(SRResources.ResourceNotFound, request.RequestUri),
		SRResources.NoRouteData));
		}
		
		routeData.RemoveOptionalRoutingParameters();
		
		// routeData.Route could be null if user adds a custom route that derives from System.Web.Routing.Route explicitly
		// and add that to the RouteCollection in the web hosted case
		var invoker = (routeData.Route == null || routeData.Route.Handler == null) ?
		_defaultInvoker : new HttpMessageInvoker(routeData.Route.Handler, disposeHandler: false);
		return invoker.SendAsync(request, cancellationToken);
		}
		}	

Step 7:
HttpMessageInvoker calls SendAsync on Route Handler object.
		public class HttpMessageInvoker : IDisposable
		{
		public virtual Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
		CancellationToken cancellationToken)
		{
		if (request == null)
		{
		throw new ArgumentNullException(nameof(request));
		}
		CheckDisposed();
		
		if (NetEventSource.IsEnabled) NetEventSource.Enter(this, request);
		
		Task<HttpResponseMessage> task = _handler.SendAsync(request, cancellationToken);
		
		if (NetEventSource.IsEnabled) NetEventSource.Exit(this, task);
		
		return task;
		}
		}	

Extra :	Link to understand how to add Per Route Handler or the global Handler.	https://www.c-sharpcorner.com/article/webapi-pipeline-revealed-a-true-practical-approach/	

Step 10:
HttpControllerDispatcher
	Role: Select the controller (returns Controller Description).
	Create the controller (Calls on ControllerDescriptor)
	Create context of Controller
		protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)
		{
		HttpControllerDescriptor controllerDescriptor = ControllerSelector.SelectController(request);
		IHttpController controller = controllerDescriptor.CreateController(request);
		controllerContext = CreateControllerContext(request, controllerDescriptor, controller);
		return await controller.ExecuteAsync(controllerContext, cancellationToken);
		}	

		private IHttpControllerSelector ControllerSelector
		{
		get
		{
		if (_controllerSelector == null)
		{
		_controllerSelector = _configuration.Services.GetHttpControllerSelector();
		}
		
		return _controllerSelector;
		}
		}	

HttpControllerDescriptor		public virtual IHttpController CreateController(HttpRequestMessage request)
		{
		if (request == null)
		{
		throw Error.ArgumentNull("request");
		}
		
		// Invoke the controller activator
		IHttpControllerActivator activator = Configuration.Services.GetHttpControllerActivator();
		IHttpController instance = activator.Create(request, this, ControllerType);
		return instance;
		}	

		//Properties
		private readonly ConcurrentDictionary<object, object> _properties = new ConcurrentDictionary<object, object>();
		private object[] _attributeCache;
		private object[] _declaredOnlyAttributeCache;
		
		public HttpConfiguration Configuration
		public string ControllerName
		public Type ControllerType
		
		public virtual Collection<IFilter> GetFilters()
		public virtual Collection<T> GetCustomAttributes<T>()
		public virtual Collection<T> GetCustomAttributes<T>(bool inherit)	

Step 16:
IHttpController -- ApiController
 	
Role:
1. Select the Action from Controller class.
2. Finds action Filters, Authentication filters and authorization filters
3. Run all filters
4. Run the action method.(run ExecuteAsync on ActionFilterResult)
5. If the action produces an exception AND an exception filter exists, the exception filter will be executed.
		public virtual Task<HttpResponseMessage> ExecuteAsync(HttpControllerContext controllerContext, CancellationToken cancellationToken)
		{
		Initialize(controllerContext);
		
		HttpControllerDescriptor controllerDescriptor = controllerContext.ControllerDescriptor;
		ServicesContainer controllerServices = controllerDescriptor.Configuration.Services;
		
		HttpActionDescriptor actionDescriptor = controllerServices.GetActionSelector().SelectAction(controllerContext);
		ActionContext.ActionDescriptor = actionDescriptor;
		if (Request != null)
		{
		Request.SetActionDescriptor(actionDescriptor);
		}	

		FilterGrouping filterGrouping = actionDescriptor.GetFilterGrouping();
		
		IActionFilter[] actionFilters = filterGrouping.ActionFilters;
		IAuthenticationFilter[] authenticationFilters = filterGrouping.AuthenticationFilters;
		IAuthorizationFilter[] authorizationFilters = filterGrouping.AuthorizationFilters;
		IExceptionFilter[] exceptionFilters = filterGrouping.ExceptionFilters;
		
		IHttpActionResult result = new ActionFilterResult(actionDescriptor.ActionBinding, ActionContext,
		controllerServices, actionFilters);
		if (authorizationFilters.Length > 0)
		{
		result = new AuthorizationFilterResult(ActionContext, authorizationFilters, result);
		}	

		if (authenticationFilters.Length > 0)
		{
		result = new AuthenticationFilterResult(ActionContext, this, authenticationFilters, result);
		}
		
		if (exceptionFilters.Length > 0)
		{
		IExceptionLogger exceptionLogger = ExceptionServices.GetLogger(controllerServices);
		IExceptionHandler exceptionHandler = ExceptionServices.GetHandler(controllerServices);
		result = new ExceptionFilterResult(ActionContext, exceptionFilters, exceptionLogger, exceptionHandler,
		result);
		}
		
		return result.ExecuteAsync(cancellationToken);
		}	

Step 19: ActionFilterResult	Role: 1. Run Model Binder 2. Run Action Invoker if find no action filter. 3. Run all action filter first then invoke action method.		
Step 21: Model Binding	Each parameter needed by the action can be bound to its value by one of three separate paths. Which path the binding system uses depends on where the value needed exists within the request.	1. f data needed for an action parameter value exists in the entity body, Web API reads the body of the request; an instance of FormatterParameterBinding will invoke the appropriate formatter classes. which bind the values to a media type (using MediaTypeFormatter). which results in a new complex type	
		2. If data needed for a parameter value exists in the URL or query string, said URL is passed into an instance of IModelBinder, which uses an IValueProvider to map values to a model. which results in a simple type.	
		3. If a custom HttpParameterBinding exists, the system uses that custom binding to build the value. which results in any kind (simple or complex) of object being mappable	
Step 32: Result Conversion	1. If the return type is already an HttpResponseMessage, we don't need to do any conversion, so pass the return on through.		
	2. If the return type is void, .NET will return an HttpResponseMessage with the status 204 No Content.		
	3. If the return type is an IHttpActionResult, call the method ExecuteAsync to create an HttpResponseMessage.	In any Web API method in which you use return Ok(); or return BadRequest(); or something similar, that return statement follows this process, rather than any of the other processes, since the return type of those actions is IHttpActionResult.	
	4. For all other types, .NET will create an HttpResponseMessage and place the serialized value of the return in the body of that message.		
Step 38: AuthenticationFilters Step 39: HttpControllerDispatcher => No job to do Step 40: HttpRoutingDispatcher => No job to do Step 41: DelegatingHandlers:At this point, the DelegatingHandler objects can really only change the response being sent (e.g. intercept certain responses and change to the appropriate HTTP status) Step 42: The final HttpResponseMessage is given to the HttpServer instance Step 43: which returns an Http response to the invoking client.			
Response Type and more exploration	These types are part of BadRequestErrorMessageResult BadRequestResult ConflictResult CreatedAtRouteNegotiatedContentResult<T> CreatedNegotiatedContentResult<T> ExceptionResult FormattedContentResult<T> InternalServerErrorResult InvalidModelStateResult JsonResult<T> NegotiatedContentResult<T> NotFoundResult OkNegotiatedContentResult<T> OkResult RedirectResult RedirectToRouteResult ResponseMessageResult StatusCodeResult UnauthorizedResult	ApiController provide some helper method to return result of type IHttpActionResult.	
	In Web API, controller class inherits from ApiController class where in ASP. NET MNC, Controller is inherited from Controller classs		
APIController adds feature to each Controller	<ApiController Feature>		
Return type of controller action	A Web API controller action can return any of the following: void HttpResponseMessage IHttpActionResult Some other type		
	Return type How Web API creates the response void : Return empty 204 (No Content) HttpResponseMessage: Convert directly to an HTTP response message. IHttpActionResult : Call ExecuteAsync to create an HttpResponseMessage, then convert to an HTTP response message. Other type : Write the serialized return value into the response body; return 200 (OK).		

HttpResponseMessage: Example		public HttpResponseMessage Get()
		{
		HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK, "value");
		response.Content = new StringContent("hello", Encoding.Unicode);
		response.Headers.CacheControl = new CacheControlHeaderValue()
		{
		MaxAge = TimeSpan.FromMinutes(20)
		};
		return response;
		}	

IHttpActionResult	The IHttpActionResult interface was introduced in Web API 2. Essentially, it defines an HttpResponseMessage factory		
advantages of using the IHttpActionResult interface	Simplifies unit testing your controllers. Moves common logic for creating HTTP responses into separate classes. Makes the intent of the controller action clearer, by hiding the low-level details of constructing the response.		

	If a controller action returns an IHttpActionResult, Web API calls the ExecuteAsync method to create an HttpResponseMessage. Then it converts the HttpResponseMessage into an HTTP response message.	public interface IHttpActionResult
		{
		Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken);
		}	

Web API Help Page	Web Api framework provides in built support of Help Page. IApiExplorer Interface provides the details of API.		https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/creating-api-help-pages
<HandsOn>			
Web API Routing Vs MVC Routing	If you are familiar with ASP.NET MVC, Web API routing is very similar to MVC routing. The main difference is that Web API uses the HTTP verb, not the URI path, to select the action. You can also use MVC-style routing in Web API.		

	To determine which action to invoke, the framework uses a routing table.
	To find the controller, Web API adds "Controller" to the value of the {controller} variable.
	To find the action, Web API looks at the HTTP verb, and then looks for an action whose name begins with that HTTP verb name. For example, with a GET request, Web API looks for an action prefixed with "Get", such as "GetContact" or "GetAllContacts". This convention applies only to GET, POST, PUT, DELETE, HEAD, OPTIONS, and PATCH verbs. You can enable other HTTP verbs by using attributes on your controller. Other placeholder variables in the route template, such as {id}, are mapped to action parameters.
		GlobalConfiguration.Configuration.Routes.MapHttpRoute("","",null);
		
		routes.MapHttpRoute(
		name: "API Default",
		routeTemplate: "api/{controller}/{id}",
		defaults: new { id = RouteParameter.Optional }
		);	

	Instead of using the naming convention for HTTP verbs, you can explicitly specify the HTTP verb for an action by decorating the action method with one of the following attributes: [HttpGet] [HttpPut] [HttpPost] [HttpDelete] [HttpHead] [HttpOptions] [HttpPatch]		

	To allow multiple HTTP verbs for an action, or to allow HTTP verbs other than GET, PUT, POST, DELETE, HEAD, OPTIONS, and PATCH, use the [AcceptVerbs] attribute, which takes a list of HTTP verbs	public class ProductsController : ApiController
		{
		[AcceptVerbs("GET", "HEAD")]
		public Product FindProduct(id) { }
		}	

Routing by Action Name	you can also create a route where the action name is included in the URI:	routes.MapHttpRoute(name: "ActionApi", routeTemplate: "api/{controller}/{action}/{id}", defaults: new { id = RouteParameter.Optional });	
	You can override the action name by using the [ActionName] attribute.	[HttpGet] [ActionName("Thumbnail")] public HttpResponseMessage GetThumbnailImage(int id);	
Non-Actions	To prevent a method from getting invoked as an action, use the [NonAction] attribute.	[NonAction] public string GetPrivateData() { ... }	
<Routing and Action Selection in ASP.NET Web API>			
Routing process	1. Matching the URI to a route template. 2. Selecting a controller. 3. Selecting an action.		
URI template	A route template looks similar to a URI path, but it can have placeholder values, indicated with curly braces.	api/{controller}/public/{category}/{id}	
	When you create a route, you can provide default values for some or all of the placeholders:	defaults: new { category = "all" }	
	You can also provide constraints, which restrict how a URI segment can match a placeholder:	constraints: new { id = @"\d+" }	
	The framework tries to match the segments in the URI path to the template. Literals in the template must match exactly. A placeholder matches any value, unless you specify constraints. The framework does not match other parts of the URI, such as the host name or the query parameters. The framework selects the first route in the route table that matches the URI.		
	There are two special placeholders: "{controller}" and "{action}". "{controller}" provides the name of the controller. "{action}" provides the name of the action. In Web API, the usual convention is to omit "{action}".		
Route Dictionary	If the framework finds a match for a URI, it creates a dictionary that contains the value for each placeholder. The keys are the placeholder names, not including the curly braces. The values are taken from the URI path or from the defaults. The dictionary is stored in the IHttpRouteData object.		
	During this route-matching phase, the special "{controller}" and "{action}" placeholders are treated just like the other placeholders. They are simply stored in the dictionary with the other values.	For the URI path "api/products", the route dictionary will contain: controller: "products" category: "all"	
	A default can have the special value RouteParameter.Optional. If a placeholder gets assigned this value, the value is not added to the route dictionary.	For "api/products/toys/123", however, the route dictionary will contain: controller: "products" category: "toys" id: "123"	
Selecting a Controller	Look in the route dictionary for the key "controller". Take the value for this key and append the string "Controller" to get the controller type name. Look for a Web API controller with this type name.	<Code Walk Trough> <DefaultHttpControllerSelector>	
Action Selection	The HTTP method of the request. The "{action}" placeholder in the route template, if present. The parameters of the actions on the controller.	<ApiControllerActionSelector>	
Which methods on the controller are considered "actions"?	1. f the name of the controller method starts with "Get", "Post", "Put", "Delete", "Head", "Options", or "Patch", then by convention the action supports that HTTP method. 2. You can specify the HTTP method with an attribute: AcceptVerbs, HttpDelete, HttpGet, HttpHead, HttpOptions, HttpPatch, HttpPost, or HttpPut.		
Parameter Bindings	A parameter binding is how Web API creates a value for a parameter. Here is the default rule for parameter binding: Simple types are taken from the URI. Complex types are taken from the request body.		
<Explore>		https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-and-action-selection#extension-points	
<Attribute Routing in ASP.NET Web API 2>			
Why Attribute Routing?	The first release of Web API used convention-based routing. In that type of routing, you define one or more route templates, which are basically parameterized strings. When the framework receives a request, it matches the URI against the route template. One advantage of convention-based routing is that templates are defined in a single place, and the routing rules are applied consistently across all controllers. Unfortunately, convention-based routing makes it hard to support certain URI patterns that are common in RESTful APIs. For example, resources often contain child resources: Customers have orders, movies have actors, books have authors, and so forth. It's natural to create URIs that reflect these relations:	/customers/1/orders	

Enabling Attribute Routing	To enable attribute routing, call MapHttpAttributeRoutes during configuration. This extension method is defined in the System.Web.Http. HttpConfigurationExtensions class.	public static class WebApiConfig {
		public static void Register(HttpConfiguration config) { // Web API routes
		config.MapHttpAttributeRoutes(); } }	

		[Route("customers/{customerId}/orders")] [HttpGet] public IEnumerable<Order> FindOrdersByCustomer(int customerId) { ... }	
	set a common prefix for an entire controller by using the [RoutePrefix] attribute:	[RoutePrefix("api/books")] public class BooksController : ApiController {.......}	

	Use a tilde (~) on the method attribute to override the route prefix:	[RoutePrefix("api/books")]
		public class BooksController : ApiController
		{
		// GET /api/authors/1/books
		[Route("~/api/authors/{authorId:int}/books")]
		public IEnumerable<Book> GetByAuthor(int authorId) { ... }
		
		// ...
		}	

Route Constraints	Route constraints let you restrict how the parameters in the route template are matched. The general syntax is "{parameter:constraint}"	[Route("users/{id:int}")] Or [Route("users/{id:int:min(1)}")] public User GetUserById(int id) { ... }	
Custom Route Constraints		https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2#custom-route-constraints	
Route Names		[Route("api/books/{id}", Name="GetBookById")] public BookDto GetBook(int id) { ...}	
Custom Media formatter		https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/media-formatters#example-creating-a-csv-media-formatter	
Accept, Content-Type, encoding, Header in http request			
JSON Media-Type Formatter	JSON formatting is provided by the JsonMediaTypeFormatter class. By default, JsonMediaTypeFormatter uses the Json.NET library to perform serialization. Json.NET is a third-party open source project		

	If you prefer, you can configure the JsonMediaTypeFormatter class to use the DataContractJsonSerializer instead of Json.NET. To do so, set the UseDataContractJsonSerializer property to true:	var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
		json.UseDataContractJsonSerializer = true;	

What Gets Serialized?	By default, all public properties and fields are included in the serialized JSON. To omit a property or field, decorate it with the JsonIgnore attribute.
	
	Read-only properties are serialized by default.
		public class Product
		{
		public string Name { get; set; }
		public decimal Price { get; set; }
		[JsonIgnore]
		public int ProductCode { get; set; } // omitted
		}	

	If you prefer an "opt-in" approach, decorate the class with the DataContract attribute. If this attribute is present, members are ignored unless they have the DataMember. You can also use DataMember to serialize private members.	[DataContract]
		public class Product
		{
		[DataMember]
		public string Name { get; set; }
		[DataMember]
		public decimal Price { get; set; }
		public int ProductCode { get; set; } // omitted by default
		}	

Indenting	To write indented JSON, set the Formatting setting to Formatting.Indented:	var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
		json.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;	

Camel Casing	To write JSON property names with camel casing, without changing your data model, set the CamelCasePropertyNamesContractResolver on the serializer:	var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
		json.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();	

XML Media-Type Formatter	XML formatting is provided by the XmlMediaTypeFormatter class. By default, XmlMediaTypeFormatter uses the DataContractSerializer class to perform serialization.
	
	If you prefer, you can configure the XmlMediaTypeFormatter to use the XmlSerializer instead of the DataContractSerializer. To do so, set the UseXmlSerializer property to true:
		var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
		xml.UseXmlSerializer = true;	

	The XmlSerializer class supports a narrower set of types than DataContractSerializer, but gives more control over the resulting XML. Consider using XmlSerializer if you need to match an existing XML schema.		
XML Serialization	Behaviors of the XML formatter using the default DataContractSerializer: All public read/write properties and fields are serialized. To omit a property or field, decorate it with the IgnoreDataMember attribute. Private and protected members are not serialized. Read-only properties are not serialized. (However, the contents of a read-only collection property are serialized.) Class and member names are written in the XML exactly as they appear in the class declaration. A default XML namespace is used.		
	If you need more control over the serialization, you can decorate the class with the DataContract attribute. When this attribute is present, the class is serialized as follows: "Opt in" approach: Properties and fields are not serialized by default. To serialize a property or field, decorate it with the DataMember attribute. To serialize a private or protected member, decorate it with the DataMember attribute. Read-only properties are not serialized. To change how the class name appears in the XML, set the Name parameter in the DataContract attribute. To change how a member name appears in the XML, set the Name parameter in the DataMember attribute. To change the XML namespace, set the Namespace parameter in the DataContract class.		

set an XML serializer for a particular type, call SetSerialize		var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
		// Use XmlSerializer for instances of type "Product":
		xml.SetSerializer<Product>(new XmlSerializer(typeof(Product)));	

Removing the JSON or XML Formatter		void ConfigureApi(HttpConfiguration config)
		{
		// Remove the JSON formatter
		config.Formatters.Remove(config.Formatters.JsonFormatter);
		// or
		// Remove the XML formatter
		config.Formatters.Remove(config.Formatters.XmlFormatter);
		}	

Handling Circular Object References	By default, the JSON and XML formatters write all objects as values. If two properties refer to the same object, or if the same object appears twice in a collection, the formatter will serialize the object twice. This is a particular problem if your object graph contains cycles, because the serializer will throw an exception when it detects a loop in the graph.		

	To preserve object references in JSON, add the following code to Application_Start method in the Global.asax file:	var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
		json.SerializerSettings.PreserveReferencesHandling =
		Newtonsoft.Json.PreserveReferencesHandling.All;	

	Now the controller action will return JSON that looks like this:	{"$id":"1","Name":"Sales","Manager":{"$id":"2","Name":"Alice","Department":{"$ref":"1"}}}	
	To preserve object references in XML, you have two options. The simpler option is to add [DataContract(IsReference=true)] to your model class. The IsReference parameter enables object references.		

	another option: Create a new type-specific DataContractSerializer instance and set preserveObjectReferences to true in the constructor. Then set this instance as a per-type serializer on the XML media-type formatter.	var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
		var dcs = new DataContractSerializer(typeof(Department), null, int.MaxValue,
		false, /* preserveObjectReferences: */ true, null);
		xml.SetSerializer<Department>(dcs);	

What is BSON?	BSON is a binary serialization format. "BSON" stands for "Binary JSON", but BSON and JSON are serialized very differently. BSON is "JSON-like", because objects are represented as name-value pairs, similar to JSON. Unlike JSON, numeric data types are stored as bytes, not strings. BSON was designed to be lightweight, easy to scan, and fast to encode/decode.		
content negotiation	the process of selecting the best representation for a given response when there are multiple representations available.		
	The primary mechanism for content negotiation in HTTP are these request headers: Accept: Which media types are acceptable for the response, such as "application/json," "application/xml," or a custom media type such as "application/vnd.example+xml" Accept-Charset: Which character sets are acceptable, such as UTF-8 or ISO 8859-1. Accept-Encoding: Which content encodings are acceptable, such as gzip. Accept-Language: The preferred natural language, such as "en-us".		
<Model Validation>			
Under-Posting	Under-posting happens when the client leaves out some properties		
Over-Posting	A client can also send more data than you expected		
Model Binding	When Web API calls a method on a controller, it must set values for the parameters, a process called binding.		
Web API uses the following rules to bind parameters	If the parameter is a "simple" type, Web API tries to get the value from the URI. Simple types include the .NET primitive types (int, bool, double, and so forth), plus TimeSpan, DateTime, Guid, decimal, and string, plus any type with a type converter that can convert from a string. (More about type converters later.) For complex types, Web API tries to read the value from the message body, using a media-type formatter.		
Using [FromUri]	To force Web API to read a complex type from the URI, add the [FromUri] attribute to the parameter.	public HttpResponseMessage Get([FromUri] GeoPoint location) { ... }	
Using [FromBody]	To force Web API to read a simple type from the request body, add the [FromBody] attribute to the parameter	public HttpResponseMessage Post([FromBody] string name) { ... }	
	At most one parameter is allowed to read from the message body. The reason for this rule is that the request body might be stored in a non-buffered stream that can only be read once.		
Type Converters	You can make Web API treat a class as a simple type (so that Web API will try to bind it from the URI) by creating a TypeConverter and providing a string conversion.		https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/parameter-binding-in-aspnet-web-api#type-converters
Parameter Binding in Web API	<Go through the request response pipeline and then explore>		https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/parameter-binding-in-aspnet-web-api#model-binders
Additional Resources for parameter binding	<Must go through all links>		https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/parameter-binding-in-aspnet-web-api#additional-resources
<Error Handling>			
<Exception Handling in ASP.NET Web API>	Consider following to handle exception in web api		

	HttpResponseException: The HttpResponseException type is a special case. This exception returns any HTTP status code that you specify in the exception constructor	Product item = repository.Get(id);
		if (item == null)
		{
		throw new HttpResponseException(HttpStatusCode.NotFound);
		}	

	For more control over the response, you can also construct the entire response message and include it with the HttpResponseException	if (item == null)
		{
		var resp = new HttpResponseMessage(HttpStatusCode.NotFound)
		{
		Content = new StringContent(string.Format("No product with ID = {0}", id)),
		ReasonPhrase = "Product ID Not Found"
		};
		throw new HttpResponseException(resp);
		}	

	Exception Filters: You can customize how Web API handles exceptions by writing an exception filter. An exception filter is executed when a controller method throws any unhandled exception that is not an HttpResponseException exception.
	Exception filters implement the System.Web.Http.Filters.IExceptionFilter interface. The simplest way to write an exception filter is to derive from the System.Web.Http.Filters.ExceptionFilterAttribute class and override the OnException method
		public class NotImplExceptionFilterAttribute : ExceptionFilterAttribute {
		public override void OnException(HttpActionExecutedContext context) {
		if (context.Exception is NotImplementedException) {
		context.Response = new HttpResponseMessage(HttpStatusCode.NotImplemented);
		}
		}
		}	

	Registering Exception Filters: By action	[NotImplExceptionFilter] public Contact GetContact(int id) {	

	By controller	[NotImplExceptionFilter]
		public class ProductsController : ApiController {
		// ... }	

	Globally	GlobalConfiguration.Configuration.Filters.Add(new ProductStore.NotImplExceptionFilterAttribute());	

	HttpError: The HttpError object provides a consistent way to return error information in the response body.
	CreateErrorResponse is an extension method defined in the System.Net.Http.HttpRequestMessageExtensions class. Internally, CreateErrorResponse creates an HttpError instance and then creates an HttpResponseMessage that contains the HttpError.
	advantage of using HttpError is that it goes through the same content-negotiation and serialization process as any other strongly-typed model.
		if (item == null)
		{
		var message = string.Format("Product with id = {0} not found", id);
		return Request.CreateErrorResponse(HttpStatusCode.NotFound, message);
		}	

IExceptionLogger and IExceptionHandler	Explore		
1.1 REST Architecture and Its contraints --Done 1.2 Web Api Request- response pipeline --Done(Code Walktrough pending) 1.3 Routing --Done 1.4 Model, Controller and Action and its role --Done 1.5 Response Type and more exploration --Done Model Binding --Pending Validation - Framework Provided support --Done 1.6 Request-Response Serialization/deserialization --Done 1.7 Content Negotiation --Done 1.8 Media Formatters --Done 1.9 Configuration --Done 1.10 Explore Framework --Done(Will be always Inprogress :)) 1.11 Advance Topics --Done(Will be always Inprogress :)) 1.12 Security --Done(Will be always Inprogress :)) 1.13 OData --Pending (No need now) 1.14 Exlore HTTP Protocol and its restriction/protocols, Header Attributes and Usage, MIME --Done 1.15 WEB API Vs WCF --Pending			
Circular reference issue in WebAPI	https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/json-and-xml-serialization#handling-circular-object-references		

Doubts + points	Solution	Example	
MVC Architecture			
MVC pattern In detail			
Why Mvc? Compare MVC over Webforms			
Asp.Net Mvc Page life cycle and compare it with Web API Request life cycle			
Role of Model, View, Controller			
RedirectToAction()	To redirect the user to specified controller		
Advantage of MVC	1. Separation of Application Logic and View Logic 2. Url is not mapped directly with File.		
Difference between HTML helpers and server controls	The primary difference between HTML helpers and server controls comes in the technical implementation: whereas server controls are full-blown classes that derive from a particular base class, HTML helpers are exposed in the form of extension methods that extend the HtmlHelper object present in ASP.NET MVC views.		
	Most server controls leverage View State in some way, whereas HTML helpers must function without it.		
What is Filters	Filters in ASP.NET Core MVC allow you to run code before or after certain stages in the request processing pipeline. Built-in filters handle tasks such as authorization (preventing access to resources a user isn't authorized for), ensuring that all requests use HTTPS, and response caching (short-circuiting the request pipeline to return a cached response). You can create custom filters to handle cross-cutting concerns for your application. Anytime you want to avoid duplicating code across actions, filters are the solution. For example, you can consolidate error handling code in a exception filter.	https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters	
	Class needs to Implement **ActionFilterAttribute** and provide implementation of **void OnActionExecuted(ActionExecutedContext filterContext)** method		
	Use of **ActionExecutedContext** parameter		
Types of Filters	**Authorization filters** run first and are used to determine whether the current user is authorized for the current request. They can short-circuit the pipeline if a request is unauthorized. **Resource filters** are the first to handle a request after authorization. They can run code before the rest of the filter pipeline, and after the rest of the pipeline has completed. They're useful to implement caching or otherwise short-circuit the filter pipeline for performance reasons. Since they run before model binding, they're useful for anything that needs to influence model binding. **Action filters** can run code immediately before and after an individual action method is called. They can be used to manipulate the arguments passed into an action and the result returned from the action. **Exception filters** are used to apply global policies to unhandled exceptions that occur before anything has been written to the response body. **Result filters** can run code immediately before and after the execution of individual action results. They run only when the action method has executed successfully and are useful for logic that must surround view or formatter execution.		

Custom validation	inherit from the ValidationAttribute, and override the IsValid method. The IsValid method accepts two parameters, the first is an object named value and the second is a ValidationContext object named validationContext. Value refers to the actual value from the field that your custom validator is validating.	public class ClassicMovieAttribute : **ValidationAttribute**, IClientModelValidator
		{
		private int _year;
		public ClassicMovieAttribute(int Year)
		{
		_year = Year;
		}
		protected override ValidationResult IsValid(object value, ValidationContext validationContext)
		{
		Movie movie = (Movie)validationContext.ObjectInstance;
		if (movie.Genre == Genre.Classic && movie.ReleaseDate.Year > _year)
		{
		return new ValidationResult(GetErrorMessage());
		}
		return ValidationResult.Success;
		}	

	Since the example works only with Movie types, a better option is to use **IValidatableObject** as shown in the following paragraph.
	
	Alternatively, this same code could be placed in the model by implementing the Validate method on the IValidatableObject interface. While custom validation attributes work well for validating individual properties, implementing IValidatableObject can be used to implement class-level validation as seen here.
		public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
		{
		if (Genre == Genre.Classic && ReleaseDate.Year > _classicYear)
		{
		yield return new ValidationResult(
		$"Classic movies must have a release year earlier than {_classicYear}.",
		new[] { "ReleaseDate" });
		}
		}	

TempData	TempData is also a dictionary derived from TempDataDictionary class and stored in short lives session and it is a string key and object value. The difference is the life cycle of the object. TempData keeps the information for the time of an HTTP Request. This mean only from one page to another. This also works with a 302/303 redirection because it's in the same HTTP Request. It helps to maintain data when you move from one controller to other controller or from one action to other action. In other words, when you redirect, "Tempdata" helps to maintain data between those redirects. It internally uses session variables. Temp data use during the current and subsequent request only means it is used when you are sure that next request will be redirecting to next view. It requires typecasting for complex data type and check for null values to avoid error. It is generally used to store only one time messages like error messages, validation messages.		
ViewBag Vs ViewData Vs TempData	1. ViewData is a dictionary of objects that is derived from ViewDataDictionary class and is accessible using strings as keys. 2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0. 3. ViewData requires typecasting for complex data type and check for null values to avoid error. 4. ViewBag doesn't require typecasting for complex data type.		
ViewBag	ViewBag can be useful when you want to transfer temporary data (which is not included in model) from the controller to the view. The viewBag is a dynamic type property of ControllerBase class which is the base class of all the controllers. ViewBag only transfers data from controller to view, not visa-versa. ViewBag values will be null if redirection occurs. The ViewBag's life only lasts during the current http request. ViewBag values will be null if redirection occurs. ViewBag is actually a wrapper around ViewData.		

ViewData	ViewData is similar to ViewBag. It is useful in transferring data from Controller to View.
	
	ViewData is a dictionary which can contain key-value pairs where each key must be string.
	ViewData and ViewBag both use the same dictionary internally. So you cannot have ViewData Key matches with the property name of ViewBag, otherwise it will throw a runtime exception.
		IList<Student> studentList = new List<Student>();
		studentList.Add(new Student(){ StudentName = "Bill" });
		ViewData["students"] = studentList;	

TempData	TempData in ASP.NET MVC can be used to store temporary data which can be used in the subsequent request. TempData will be cleared out after the completion of a subsequent request. TempData is useful when you want to transfer non-sensitive data from one action method to another action method of the same or a different controller as well as redirects. Call TempData.Keep() to retain TempData values in a third consecutive request. TemData is a TempDataDictionary type. TempData internaly use Session to store the data.	TempData["name"] = "Test data";	
HTML Helper method	HTML helper are used to create/render HTML Content.They are Extension method.		
Helper Methods	Action : Process child action method of controller. ActionLink : Generates anchor tag. Display: Lets say we have a list of object and we want to render single property of all objects of list as string in HTML then 'Display' will be used. Single method call will render all object's property. DisplayName: Display the property Name as plain text in HTML. DisplayText: DisplayName Vs. Label: Directtly renders text into HTML Where Lable creates 'lable' tag into html Editor Vs. Textbox : bOTH GENERATE TEXTBOX IF TEMPLATE IS NOT PROVIDED. In case of Provided template, It will render the control as specifed in template file. Name: Value:	@Html.TextBox("firstname") @Html.TextBox("firstname", "John") <input id="firstname" name="firstname" type="text" value="John" /> @Html.Label("Text Content",new { id = "id-element", name = "name-element", size = 10, @class = "css-class" }) The second parameter is the htmlAttributes, so, you can add any html attribute you want as a property of this anonymous object. To set HTML attributes, use the following overloaded version. Notice that, we are passing HTML attributes (style & title) as an anonymous type. @Html.TextBox("firstname", "John", new { style = "background-color:Red; color: White; font-weight:bold", title="Please enter your first name" }) Some of the html attributes, are reserved keywords. Examples include class, readonly etc. To use these attributes, use "@" symbol as shown below.	
Textbox vs. TextboxFor	Textbox is not strongly type method where TextboxType is strongly Type.End Result will be same.Textbox wont give compile time error but TextboxFor will give. If View is not strogly bind, TextboxFor will be useless.		

To generate a dropdownlist : using hard coded values	From the database table
	public ActionResult Index()
	{
	SampleDBContext db = new SampleDBContext();
	// Retrieve departments, and build SelectList
	ViewBag.Departments = new SelectList(db.Departments, "Id", "Name");
	return View();
	}
	@Html.DropDownList("Departments", "Select Department")
		overloaded version of "DropDownList" html helper.
		DropDownList(string name, IEnumerable<SelectListItem> selectList, string optionLabel)
		
		@Html.DropDownList("Departments", new List<SelectListItem>
		{
		new SelectListItem { Text = "IT", Value = "1", Selected=true},
		new SelectListItem { Text = "HR", Value = "2"},
		new SelectListItem { Text = "Payroll", Value = "3"}
		}, "Select Department")	

Model Binding	Mapping between incoming request data and application models is handled by model binders.		
How model binding works	When MVC receives an HTTP request, it routes it to a specific action method of a controller. It determines which action method to run based on what is in the route data, then it binds values from the HTTP request to that action method's parameters. MVC will try to bind request data to the action parameters by name. MVC will look for values for each parameter using the parameter name and the names of its public settable properties. MVC will bind data from various parts of the request and it does so in a set order	1. Form values: These are form values that go in the HTTP request using the POST method. (including jQuery POST requests). 2. Route values: The set of route values provided by Routing 3. Query strings: The query string part of the URI. In order for binding to happen the class must have a public default constructor and member to be bound must be public writable properties. When a parameter is bound, model binding stops looking for values with that name and it moves on to bind the next parameter. If binding fails, MVC does not throw an error. You can query for model state errors by checking the **ModelState.IsValid** property.	
Customize model binding behavior with attributes	MVC contains several attributes that you can use to direct its default model binding behavior to a different source. For example, you can specify whether binding is required for a property, or if it should never happen at all by using the [BindRequired] or [BindNever] attributes. Alternatively, you can override the default data source, and specify the model binder's data source. Below is a list of model binding attributes:	[BindRequired]: This attribute adds a model state error if binding cannot occur. [BindNever]: Tells the model binder to never bind to this parameter. [FromHeader], [FromQuery], [FromRoute], [FromForm]: Use these to specify the exact binding source you want to apply. [FromServices]: This attribute uses dependency injection to bind parameters from services. [FromBody]: Use the configured formatters to bind data from the request body. The formatter is selected based on content type of the request. [ModelBinder]: Used to override the default model binder, binding source and name.	
Model Validation		https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation	
HTML Helper method	HTML helper are used to create/render HTML Content.They are Extension method.		
Helper Methods	Action : Process child action method of controller. ActionLink : Generates anchor tag. Display: Lets say we have a list of object and we want to render single property of all objects of list as string in HTML then 'Display' will be used. Single method call will render all object's property. DisplayName: Display the property Name as plain text in HTML. DisplayText: DisplayName Vs. Label: Directtly renders text into HTML Where Lable creates 'lable' tag into html Editor Vs. Textbox : bOTH GENERATE TEXTBOX IF TEMPLATE IS NOT PROVIDED. In case of Provided template, It will render the control as specifed in template file. Name: Value:	@Html.TextBox("firstname") @Html.TextBox("firstname", "John") <input id="firstname" name="firstname" type="text" value="John" /> @Html.Label("Text Content",new { id = "id-element", name = "name-element", size = 10, @class = "css-class" }) The second parameter is the htmlAttributes, so, you can add any html attribute you want as a property of this anonymous object. To set HTML attributes, use the following overloaded version. Notice that, we are passing HTML attributes (style & title) as an anonymous type. @Html.TextBox("firstname", "John", new { style = "background-color:Red; color: White; font-weight:bold", title="Please enter your first name" }) Some of the html attributes, are reserved keywords. Examples include class, readonly etc. To use these attributes, use "@" symbol as shown below.	
HTML Helpers are categorized into three types	1. Inline HTML Helpers 2. Built-in HTML Helpers 3. Custom HTML Helpers		
Built-in HTML Helpers are further divided into three categories:	1. Standard HTML Helpers 2. Strongly Typed HTML Helpers 3. Templated HTML Helpers		
Standard HTML Helpers	Standard HTML Helpers are used to render the most common type of HTML controls like TextBox, DropDown, Radio buttons, Checkbox etc. Extension methods of HTML Helper classes have several overloaded versions. We can use any one according to our requirement.		
Strongly Typed HTML Helpers	Just like Standard Helper, we have several strongly typed methods. Strongly Typed Helper requires lambda expressions.	Html.TextBoxFor(), Html.TextAreaFor(), Html.DropDownListFor(), Html. CheckboxFor(), Html.RadioButtonFor(), Html.ListBoxFor(), Html.PasswordFor(), Html.HiddenFor(), Html.LabelFor(), etc	
Textbox vs. TextboxFor	Textbox is not strongly type method where TextboxType is strongly Type.End Result will be same.Textbox wont give compile time error but TextboxFor will give. If View is not strogly bind, TextboxFor will be useless.		
Template Helper Method	These methods are very flexible and generate the HTML element based on the properties of the model class. We have already seen an EditorFor Helper method in the previous example, which generates TextArea element because we have declared MultiLine Datatype on Address property. Display, DisplayFor, Editor, and EditorFor are the examples of Template Helper method.		https://dzone.com/articles/working-with-built-in-html-helper-classes-in-aspne
Custom Html Helper			
Best way to bind Dropdown Item source			
Dropdown filter runtime	Can use ajax to bind the dropdown options		
few in built behaviour and its variant of controller class	1. PartialView() 2. Redirect() 3. RedirectToAction() 4. RedirectToActionPermanent() 5. RedirectToRoute() 6. RedirectToRoutePermanent() 7. ValidateModel() 8. View() 9. ModelState => Isvalid & AddModelError 10. HttpContext 11. Request 12. Response 13. RouteData 14. Session 15. Url		
What is Model Binding	MVC framework converts the http request values (from query string or form collection) to action method parameters. These parameters can be of primitive type or complex type		https://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc
Default Model Binding algorithm	Binding to Primitive type: HttpGET request embeds data into a query string. MVC framework automatically converts a query string to the action method parameters. For example, the query string "id" in the following GET request would automatically be mapped to the id parameter of the Edit() action method.		
	Binding to Complex type: Model binding also works on complex types. Model binding in MVC framework automatically converts form field data of HttpPOST request to the properties of a complex type parameter of an action method.		
	FormCollection: You can also include FormCollection type parameter in the action method instead of complex type, to retrieve all the values from view form fields		

Bind Attribute	ASP.NET MVC framework also enables you to specify which properties of a model class you want to bind. The [Bind] attribute will let you specify the exact properties a model binder should include or exclude in binding.	[HttpPost]
		public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)
		{
		var name = std.StudentName;
		}	

Action Name Attribute	ActionName attribute allows us to specify a different action name than the method name. View name also should be the same as ActionName.		
UpdateModel and TryUpdateModel	Functions UpdateModel and TryUpdateModel are used to update the model with the form values and perform the validations.		
UpdateModel vs TryUpdateModel	UpdateModel will throw exception if it fails to update the model where TryUpdateModel won't.		
UpdateModel<Interface>	We can prevent unintented update into model using interface.	UpdateModel<IEmployee>(employee); employee will be updated with the field defined in Iemployee interface.	
FormCollection	Form collection is one way to pass a value from a view to the controller		
Required Attribute	Specifies that a data field value is required.		
ModelState			https://exceptionnotfound.net/asp-net-mvc-demystified-modelstate/
ModelState.AddModelError("FieldID","Error Message")			

MetadataType	Specifies the metadata class to associate with a data model class.	[MetadataType(typeof(CustomerMetaData))]
		public partial class Customer
		{
		}
		
		public class CustomerMetaData
		{
		// Apply RequiredAttribute
		[Required(ErrorMessage = "Title is required.")]
		public object Title;
		}	

Scaffold Attribute	<TBD>		
DataType attribute	DataType.EmailAddress : DataType.Currency: DataType.Url DataType.Time DataType.Date		
DisplayColumn Attribute	Display single property specified in attribute for complex datatype.		

What is View Engine	View Engine is responsible for rendering the view into html form to the browser.		
Razor Vs. Aspx View Engine		http://www.dotnet-tricks.com/Tutorial/mvc/91JM151212-Difference-Between-Razor-View-Engine-and-ASPX-View-Engine.html	
Custom View Engine			
Order of view file to be searched to render.	~/Views/Employee/Index.aspx ~/Views/Employee/Index.ascx ~/Views/Shared/Index.aspx ~/Views/Shared/Index.ascx ~/Views/Employee/Index.cshtml ~/Views/Employee/Index.vbhtml ~/Views/Shared/Index.cshtml ~/Views/Shared/Index.vbhtml		
EditorTemplates	Both contains templates means raw html control define somewhere by framework and rendered in UI when DisplayFor/EditorFor will be called.behaviour of system defined template can be chaned by applying metadata to property.U can override control behaviour	When we use DisplayFor/EditorFor (Premitive Data type), It uses Predefined html control.For e.g. string type property will be rendered as textbox.(If EditorFor is used).If Property contains metadata then behaviour will be changed.for E.g. readoOnly will b	
DisplayTemplates			
UIHint	UIHintAttribute Specifies the template or user control that Dynamic Data uses to display a data field. If you annotate a property with UIHint attribute and use EditorFor or DisplayFor inside your views, ASP.NET MVC framework will look for the specified template which you specified through UIHintAttribute. The directories it looks for is: For EditorFor: ~/Views/Shared/EditorTemplates ~/Views/Controller_Name/EditorTemplates For DisplayFor: ~/Views/Shared/DisplayTemplates ~/Views/Controller_Name/DisplayTemplates		
@ViewData.ModelMetadata			http://csharp-video-tutorials.blogspot.com/2013/06/part-46-accessing-model-metadata-from.html
@ViewData.TemplateInfo.FormattedModelValue			
@ViewData.TemplateInfo			
@: or <text> element	To inform that swtich to literal from c#.		
@* ..... *@	Commeting in razor page		
@@	Escape characher		

_ViewStart	Conatains the Layout property value.it will be the file path of master file.
	If ViewStart is in Shared Folder then it will be applied to all views. If it is in specific view folder then it will be applied to those views only
		@{
		  Layout = "~/Views/Shared/_v1.cshtml";
		}	

	Can overwrite layout path by specifing it in each view file.		
	can also specify layout file path in View() function in controller.		
Named Section	Part 61 @section : To define the named section in view. RenderSection : To refer the named section in layout IsSectionDefined : to check whether section is defined or not.		http://csharp-video-tutorials.blogspot.com/2013/07/part-61-named-sections-in-layout-files.html

Handle Error: Part 72	Handle Error is used to display friendly error message for unhandled errors to UI.	<customErrors mode="On">
		  <error statusCode="404" redirect="~/Error/NotFound"/>
		</customErrors>
		
		filters.Add(new HandleErrorAttribute());	

OutputCache : Part 73	Output chache is used to cache content returned by Controller, next time it doesnt require to generate it again.	Keep [OutputCache] attribute to action method.	

Cache Profile : Part 74	cache settings can be specified in web.config file using cache profiles.
	The cache settings are now read from one central location i.e from the web.config file.
	The advantage of using cache profiles is that
	1. You have one place to change the cache settings. Mantainability is much easier.
	2. Since the changes are done in web.config, we need not build and redeploy the application.
		<system.web>
		  <caching>
		    <outputCacheSettings>
		      <outputCacheProfiles>
		        <clear/>
		        <add name="1MinuteCache" duration="60" varyByParam="none"/>
		      </outputCacheProfiles>
		    </outputCacheSettings>
		  </caching>
		</system.web>
		[OutputCache(CacheProfile = "1MinuteCache")]
		public ActionResult Index()
		{
		  return View(db.Employees.ToList());
		}	

Require Https: Part 75	Forces unsecured http request to resent over https.	[RequireHttps]	
ValidateInput : Part 76	It is used to enable or disable request validation. By default, request validation is enable and it doesn't allow to submit any html or script to prevent cross site scripting.	[ValidateInput]	
Filter In Details	There may be circumstances where you want to execute some logic before or after an action method executes. ASP.NET MVC provides filters for this purpose.		
	Four types of Filter : Authorization filters : Implements IAuthorizationFilter Action filters : Implement IActionFilter Result filters - Implement IResultFilter. Exception filters - Implement IExceptionFilter	Built In Filter: Authorization filters : [Authorize], [RequireHttps] Action filters : Result filters : OutputCache Exception filters : [HandleError]	

Custom Filters : Part 77	Custom filter class can be created by implementing FilterAttribute class and corresponding interface.	class MyErrorHandler : FilterAttribute, IExceptionFilter
		{
		  public override void IExceptionFilter.OnException(ExceptionContext filterContext)
		  {
		    Log(filterContext.Exception);
		    base.OnException(filterContext);
		  }
		
		  private void Log(Exception exception)
		  {
		    //log exception here..
		  }
		}
			https://www.tutorialsteacher.com/mvc/filters-in-asp.net-mvc

Action Result : Part 78	Action method can return a wide range of object for e.g. ViewResult PartialViewResult JsonResult RedirectResult etc.		Subtype of ActionResult :https://msdn.microsoft.com/en-us/library/system.web.mvc.actionresult%28v=vs.118%29.aspx
Area : Part 79	Need to check in interet. Expolaration required.		
StringLength : Part 80	Attribute will enforce the input to be of specified length. It will not enforce of mandatory check.		
Range: Part 81	Will enforce that input should be in range. There are many overloaded version of range.		

CustomValidation : Part 82	If we want to add our own custom validation attribute, then class should inherit 'ValidationAttribute' class and must implement 'IsValid' method.	public class FutureDateAttribute :ValidationAttribute
		  {
		    public override bool IsValid(object value)
		    {
		      return (DateTime)value >= DateTime.Now;
		    }
		  }	

Regular Expression : Part 83	We can apply the Regular expression to restrict the user input		
Compare : Part 84	To compare two fields of user input, Compare attribute is used.		
Part 85 - Enable client side validation	Add Following file reference in same order. <script src="~/Scripts/jquery-1.7.1.min.js" type="text/javascript"></script> <script src="~/Scripts/jquery.validate.min.js" type="text/javascript"></script> <script src="~/Scripts/jquery.validate.unobtrusive.min	Enable ClientValidation and UnobtrusiveJavaScript in web.config file. <appSettings> <add key="ClientValidationEnabled" value="true" /> <add key="UnobtrusiveJavaScriptEnabled" value="true" /> </appSettings>	
Part 86 Validation Summary	To show all validation summary to one location.	@Html.ValidationSummary(false, "Please check following errors", new { @class = "alert alert-danger" })	
Part 87 : Unobtrusive JavaScript	Unobtrusive JavaScript, is a JavaScript that is separated from the web site’s html markup. There are several benefits of using Unobtrusive JavaScript. Separation of concerns i.e the HTML markup is now clean without any traces of javascript. Page load time		

Part 89 - Remote validation	Sometimes, to check if a field value is valid, we may need to make a database call. A classic example of this is the user registration page. To register a user, we need a unique username. So, to check, if the username is not taken already, we have to make	//Controller Action Method:
		public JsonResult IsUserNameAvailable(string UserName)
		{
		  return Json(!db.Users.Any(x => x.UserName == UserName),
		                JsonRequestBehavior.AllowGet);
		}
		Model Property and its attribute	

Part 90 - Remote validation when javascript is disabled	Validate when data is posted back to controller.		
Part 91 - Create a custom remote attribute and override IsValid() method			http://csharp-video-tutorials.blogspot.in/2013/09/part-91-create-custom-remote-attribute.html

part 92 Ajax with asp.net mvc	ASP.NET AJAX enables a Web application to retrieve data from the server asynchronously and to update portions of the existing page.	//Code to call the ajax method
		@Ajax.ActionLink("All", "All",
		  new AjaxOptions
		  {
		    HttpMethod = "GET", // HttpMethod to use, GET or POST
		    UpdateTargetId = "divStudents", // ID of the HTML element to update
		    InsertionMode = Insertion	

Part 94 & 95 & 96 option of AjaxOption class	1.HttpMethod : set the method type of ajax call 2.UpdateTargetId : set the id og html tag where ajax return data wil be binded. 3.InsertionMode : will specify whether to replace the previous called data or append it. possible options are 'Replace','Ins		
Part 97 Autocomplete feature			http://csharp-video-tutorials.blogspot.in/2013/09/part-97-implement-autocomplete-textbox.html
Peek and keep method	But one can persist data in TempData object even after request completion with the help of Keep() or Peek() method.		http://sandeep-tada.blogspot.in/2014/07/use-tempdata-keep-or-peek-method-to.html
TempData	TempData used to transfer data between controllers or between actions. There is one point to note that TempData is only work during the current and subsequent request and it is generally used to store one time message.		
TempData.Keep():	TempData.Keep() method keep value in TemData object at the end of current request. There are two overloaded keep methods available with TempData: void Keep(): This method make ensures that all the items in TempData are not destroyed on current request c		
TempData.Peek(string key):	It returns an object that contains the element that is associated with the specified key, without marking the key for deletion.		
Keep() Vs. Peek()	To read and retain the value with Keep one need to do two request, i.e. first read the value and in next statement call Keep method to retain value.	With the help of Peek method one can do both operation in a single statement i.e. access as well retain value.	
	Keep method can be overloaded, i.e. one can keep all items or pass the key to retain specific item.	There is no overloaded method in case of Peek method.	
	RedirectResult and RedirectToRouteResult internally calls Keep method to retain items	Peek method is not called internally with any of ActionResult.	
@Html.EditorForModel() and @Html.DisplayForModel() , @Html.Editor @Html.EditorFor	Editor() or EditorFor() extension method generates html elements based on the data type of the model object's property. EditorFor() method is a strongly typed method. It requires the lambda expression to specify a property of the model object.	Property DataType ==> Html Element string ==> <input type="text" > int ==> <input type="number" > decimal, float ==> <input type="text" > boolean ==> <input type="checkbox" > Enum ==> <input type="text" > DateTime ==> <input type="datetime" >	
public string Content(string contentPath) method in UrlHelper Class	Converts a virtual (relative) path to an application absolute path.		
View Compilation	1. Open MVC project file using a notepad. Project files have the extension of .csproj or . vbproj 2. Search for MvcBuildViews under PropertyGroup. MvcBuildViews is false by default. Turn this to true as shown below. <MvcBuildViews>true</MvcBuildViews>		
Partial Views @Html.Partial("_Employee", item)	@Html.Partial() helper method renders the specified partial view. It accept partial view name as a string parameter and returns MvcHtmlString. It returns html string so you have a chance of modifing the html before rendering.		
html.Renderpartial	The RenderPartial helper method is same as the Partial method except that it returns void and writes resulted html of a specified partial view into a http response stream directly. Partial view is a reusable view, which can be used as a child view in multiple other views. Partial view can be rendered using Html.Partial(), Html.RenderPartial() or Html. RenderAction() method.		
html.Renderpartial and Difference between html.partial and html.renderpartial	1. The return type of "RenderPartial" is void, where as "Partial" returns "MvcHtmlString" 2. Syntax for invoking Partial() and RenderPartial() methods in Razor views @Html.Partial("PartialViewName") { Html.RenderPartial("PartialViewName"); }		
	The main difference is that "RenderPartial()" returns void and the output will be written directly to the output stream, where as the "Partial()" method returns MvcHtmlString, which can be assigned to a variable and manipulate it if required. So, when there is a need to assign the output to a variable for manipulating it, then use Partial(), else use RenderPartial(). From a performance perspective, rendering directly to the output stream is better. RenderPartial() does exactly the same thing and is better for performance over Partial().		
T4 Templates	T4 stands for Text Template Transformation Toolkit and are used by visual studio to generate code when you add a view or a controller.		
What XSS is.. Example of it and ways to do it	XSS is cross site scripting. User inject scripts through user input to change the behaviour of the application.		
framework support to prevent it ([ValidateInput(false)] ,HTML Encoding)	ValidateInput is by default enabled to prevent it.		
HttpUtility.HtmlEncode("") To Encode the string	If it disabled then user input can be encoded using HtmlEncode Method.		
@RenderBody() and Layout	The RenderBody method resides in the master page, or in Razor this is commonly referred to as the Layout page. There can only be one RenderBody method per Layout page. If you're from Web Forms world, the easiest way to think of RenderBody is it's like the ContentPlaceHolder server control. The RenderBody method indicates where view templates that are based on this master layout file should "fill in" the body content.		
@RenderSection()	Layout pages also have the concept of sections. A layout page can only contain one RenderBody method, but can have multiple sections. To create a section you use the RenderSection method. The difference between RenderSection and RenderPage is RenderPage reads the content from a file, whereas RenderSection runs code blocks you define in your content pages.		
@Html.Action() and Html.RenderAction() for [ChildActionOnly]	The RenderAction helper method invokes a specified controller and action and renders the result as a partial view. The specified Action method should return PartialViewResult using the Partial() method.	https://www.c-sharpcorner.com/UploadFile/97fc7a/child-action-methods-in-Asp-Net-mvc-4/	
RouteTable => Will have Routes Property of type RouteCollection RouteCollection is inherited from Collection<RouteBase> public RouteData GetRouteData(HttpContextBase httpContext); in RouteCollection UrlRoutingModule MVCHandler => HttpHandler RequestContext DefaultControllerFactory ControllerContext	https://www.c-sharpcorner.com/UploadFile/00a8b7/Asp-Net-mvc-life-cycle/		
RouteTable have one static property called Routes of Type RouteCollection RouteCollection manages to hold all routes of type collection RouteData. all Add Remove funcitons and Return RouteDate based on httpRequest considering all form data public RouteData GetRouteData(HttpContextBase httpContext); in RouteCollection RouteCollection is inherited from Collection<RouteBase>			
UrlRoutingModule Does three things 1. RouteData routeData = RouteCollection.GetRouteData(context); => Find the matching RouteData from RouteCollection 2. RequestContext requestContext = new RequestContext(context, routeData); // Dev10 766875 Adding RouteData to HttpContext context.Request.RequestContext = requestContext; => Converting HttpContext to RequestContext 3. context.RemapHandler(httpHandler); => Working on.			
MVC Hander Implements IHTTPAsyncHandler which has ProcessRequest() method So ProcessRequest() method will be called ProcessRequest() will call ProcessRequestInit() = > ProcessRequestInit will create a ControllerFactory and a Controller and Next, a ControllerContext object is constructed from the RequestContext and the controller using the method GetContollerInstance().			
Our Controller inherits from the Controller class that inherits from ControllerBase that implements the Icontroller interface. The Icontroller interface has an Execute() abstract method that is implemented in the ControllerBase class. so Execute() method of ControllerBase will be called. Initialize() in Execute()==> The ViewBag, ViewData, TempData and so on properties of the ControllerBase class is initialized. Execute() will call ExecuteCore() method defined in Controller() The Execute() method of the ControllerBase class is executed that calls the ExecuteCore() abstract method. ExecuteCore() is implemented in the Controller class. ExcuteCore will does 2 things 1. ActionInvoker.InvokeAction(ControllerContext, actionName) 2. PossiblySaveTempData(); 1st point in details => This builds a list of parameters from the request. This list of parameters are ed as method parameters to the ActionMethod that is executed. Finally Action method will be called.			
InvokeAction()			
ExecuteResult() of ActionResult			

Compatibility of ASP.NET Web Forms and MVC		https://docs.microsoft.com/en-us/previous-versions/aspnet/dd381619(v=vs.98)	
Understanding the MVC Application Execution Process		https://docs.microsoft.com/en-us/previous-versions/aspnet/dd381612%28v%3dvs.100%29	
Difference between html helper and Asp.Ner server control	View state will be supported in ASP.NET control and in helper methods do not have viewstate		
Any scenario where I have implemented custom filter			
Implement all html controls with HTML helper method	<RadioButton or checkbox is an issue I guess>		
Custom authentication implementation			
Better solution to pass dropdown list source to view			
An example of EditorFor and DisplayFor			

Concepts	Description	Example	
ASP pages consist of following elements.	Directives -Code Declaration blocks -Code render blocks -Asp.net Server controls -html tags		
Directives	Behavior of the asp pages can be controlled by the directives. • @Page : The @Page directive enables you to specify attributes and values for an ASP.NET Page to be used when the page is parsed and compiled • @Master : The @Master directive belongs to Master Pages that is . master files. • @Control : The @Control directive is used when we build an ASP.NET user controls. • @Register : The @Register directive associates aliases with namespaces and class names for notation in custom server control syntax. • @Reference : The @Reference directive declares that another ASP. NET page or user control should be complied along with the current page or user control. • @PreviousPageType : The @PreviousPageType is a new directive makes excellence in ASP.NET 3.5 pages. The concept of cross-page posting between ASP.NET pages is achieved by this directive • @OutputCache : The @OutputCache directive controls the output caching policies of the ASP.NET page or user control. • @Import : The @Import directive allows you to specify any namespaces to the imported to the ASP.NET pages or user controls • @Implements : The @Implements directive gets the ASP.NET page to implement a specified .NET framework interface. • @Assembly : The @Assembly directive is used to make your ASP.NET page aware of external components • @MasterType : To access members of a specific master page from a content page you can page, create a strongly typed reference to the master page by creating a @MasterType directive.	<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="LocalInvoice.aspx.cs" MasterPageFile="~/MasterPages/Edge.Master" ClientIDMode="Static" Inherits="SalesEDGE.Retail.Web.Invoicing.LocalInvoice" %> <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="InvoicingTab.ascx.cs" Inherits="SalesEDGE.Retail.Web.Invoicing.User_Controls.InvoicingTab" ClientIDMode="Static" %> <%@ Register Src="~/Invoicing/UserControls/InvoicingTab.ascx" TagName="InvoicingTabControl" TagPrefix="uc1" %> <uc1:InvoicingTabControl ID="InvoicingTabControl" runat="server" /> <%@ Import Namespace="SalesEDGE.Common.Utilities" %>	https://www.c-sharpcorner.com/UploadFile/0c1bb2/directives-in-Asp-Net-web-page428/
Page directive	• Language : It defines the language used for any inline rendering and script blocks. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET. • CodeFile : It specifies the code-behind file with which the page is associated. • ClassName : It specifies the name of the class that is bound to the page when the page is compiled. • Theme : It applies the specified theme to the page using ASP.NET2.0 themes feature. • MasterPageFile : It specifies the location of the MasterPage file to be used with the current ASP.NET page. • EnableViewState: It indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true. • Inherits: It specifies a code-behind class for the page to inherit. This can be any class derived from the Page class. • AutoEventWireup		
View state	View state is a repository in an ASP.NET page that can store values that have to be retained during postback. The page framework uses view state to persist control settings between postbacks. You can use view state in your own applications to do the following: •Keep values between postbacks without storing them in session state or in a user profile. •Store the values of page or control properties that you define. •Create a custom view state provider that lets you store view state information in a SQL Server database or in another data store.		

AutoEventWireup	Gets or sets a value indicating whether events for ASP.NET pages are automatically connected to event-handling functions.
	
	When AutoEventWireup is true, ASP.NET does not require that you explicitly bind event handlers to a page event such as Load.
	When AutoEventWireup is false, you must explicitly bind the event to a method. For example, if you have a Page_Load method in the code for a page, the method will be called in response to the Load event only if you write code like this..-->
		public partial class AutoEventWireupExample : System.Web.UI.Page
		{
		protected void Page_Load(object sender, System.EventArgs e)
		{
		Response.Write("Executing Page_Load");
		}
		override protected void OnInit(EventArgs e)
		{
		this.Load += new System.EventHandler(this.Page_Load);
		}
		}
			http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.autoeventwireup(v=vs.110).aspx

AutoPostBack	Gets or sets a value indicating whether a postback to the server automatically occurs when the user changes the list selection		
View State	HTTP is a stateless protocol so it wont retrain the state of control during the postback. ViewState is used to retain the value of control. ASP.Net internally implement the viewstate to retain the value for the asp controls where html control won't retrain the values by default. Asp.net saves all the view state information (values of all controls) in hidden control named __ViewState.	ViewState['ControlValue'] = 'value' <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTUyMDk1NTY2MGRk/8Lhq+70GXP8aAC03ELpvjoe+K4K+l4+qSiW7SdJ+TY=" />	
ViewState:	1. ViewState of a webform is available only with in that webform 2. ViewState is stored on the page in a hidden field called _ViewState. Because of this, the ViewState, will be lost, if you navigate away from the page, or if the browser is closed. 3. ViewState is used by all asp.net controls to retain their state across postback		
Session State:	1. Session state variables are available across all pages, but only for a given single session. Session variables are like single-user global data. 2. Session state variables are stored on the web server. 3. SessionState variables are cleared, when the user session times out. The default is 20 minutes. This is configurable in web.config		
Application State:	1. Application State variables are available across all pages and across all sessions. Application State variables are like multi-user global data. 2. Application State variables are stored on the web server. 3. Application State variables are cleared, when the process hosting the application is restarted.		

Cookies	Can store data at client machine.
	Used to identify the User.
	Can use when state is not so important. By default cookies will be deleted when browser will be closed. If we set the expiry. It wont delete.
		//First Way
		HttpCookie StudentCookies = new HttpCookie("StudentCookies");
		StudentCookies.Value = TextBox1.Text;
		StudentCookies.Expires = DateTime.Now.AddHours(1);
		Response.Cookies.Add(StudentCookies);	

	Disadvantage : User can delete the cookies
	can disable the cookies in browser.
		//Second Way
		Response.Cookies["StudentCookies"].Value = TextBox1.Text;
		Response.Cookies["StudentCookies"].Expires = DateTime.Now.AddDays(1);	

		//Writing Multiple values in single cookie
		Response.Cookies["StudentCookies"]["RollNumber"] = TextBox1.Text;
		Response.Cookies["StudentCookies"]["FirstName"] = "Abhimanyu";
		Response.Cookies["StudentCookies"]["MiddleName"] = "Kumar";
		Response.Cookies["StudentCookies"]["LastName"] = "Vatsa";
		Response.Cookies["StudentCookies"]["TotalMarks"] = "499";
		Response.Cookies["StudentCookies"].Expires = DateTime.Now.AddDays(1);	

		Request.Cookies["StudentCookies"].Value;	
Viewstate encryption at page level	add attribute "ViewStateEncryptionMode" to the page directive.		
Viewstate encryption at whole site	see the below code.	<configuration> <system.web><pages ViewStateEncryptionMode="always" /></system.web> </configuration>	
What is the use of EnableViewState/ViewStateMode in page directive.	??	??	
IsPostBack	Gets a value that indicates whether the page is being rendered for the first time or is being loaded in response to a postback. A postback is a request sent from a client to server from the same page, user is already working with." ASP.NET was introduced with a mechanism to post an HTTP POST request back to the same page. It's basically posting a complete page back to server (i.e. sending all of its data) on same page. So, the whole page is refreshed.		
IsCallBack	A callback is generally a call for execution of a function after another function has completed." But if we try to differentiate it from a postback then we can say: It's a call made to the server to receive specific data instead of whole page refresh like a postback. In ASP.NET, its achieved using AJAX, that makes a call to server and updating a part of the page with specific data received.		
Difference between IsCallBack and IsPostBack	IsPostBack is true when the page is posted via a form method postback is when the form is posted back to itself, either by clicking a submit button or through JavaScript (like AutoPostback controls) ViewState is updated during a Postback IsCallBack is true when the page has been called back from an AJAX call. A callback does not refresh the currently viewed page (i.e. does not redraw the page). ViewState is not updated during a callback		
IsCrossPagePostBack	Gets a value indicating whether the page is involved in a cross-page postback or not.		

Concepts	Description	Example
Configuration Files	Web.config machine.config	Machine.Config This is automatically installed when you install Visual Studio. Net. This is also called machine level configuration file. Only one machine.config file exists on a server. This file is at the highest level in the configuration hierarchy.
		Web.Config This is automatically created when you create an ASP.Net web application project. This is also called application level configuration file. This file inherits setting from the machine.config
Server.transfer Vs Response.Redirecd	The main difference between them is who does the transfer. In "response.redirect" the transfer is done by the browser while in "server.transfer" it's done by the server. 1.User sends a request to an ASP.NET page.the request is sent to "WebForm1" and we would like to navigate to "Webform2". 2.Server starts executing "Webform1" and the life cycle of the page starts. But before the complete life cycle of the page, "Server.transfer" happens to "WebForm2". 3."Webform2" page object is created, full page life cycle is executed and output HTML response is then sent to the browser. Note: One important point to note here is the URL is not changed to the target page. If you have sent request from "Webform1.aspx" to redirect to "WebForm2.aspx" on the browser URL you will still see "WebForm1.aspx".	While in "Response.Redirect" following is the sequence of events for navigation:- 1.Client (browser) sends a request to a page. the request is sent to "WebForm1" and we would like to navigate to "Webform2". Life cycle of "Webform1" starts executing. But in between of the life cycle "Response. Redirect" happens. 2.Now rather than server doing a redirect , he sends a HTTP 302 command to the browser. This command tells the browser that he has to initiate a GET request to "Webform2.aspx" page. 3.Browser interprets the 302 command and sends a GET request for "Webform2.aspx". In this case you will the URL's are changed as per redirection. So if you have redirected to "Webform2.aspx" then on the browser URL you should see "WebForm2.aspx". Use "Server.Transfer" when you want to navigate pages which reside on the same server, use "Response.Redirect" when you want to navigate between pages which resides on different server and domain.
What is importance of "preserveForm" flag in "Server. Transfer"?	"Server.Transfer" helps to redirect from one page to other page. If you wish to pass query string and form data of the first page to the target page during this redirection you need to set "preserveForm" to "true" as shown in the below code.	Server.Transfer("Webform2.aspx",true);
Response.Redirect(URL,true) vsResponse.Redirect(URL,false) ?	Response.Redirect(URL,false) :- Client is redirected to a new page and the current page on the server will keep processing ahead. Response.Redirect(URL,true) :- Client is redirected to a new page but the processing of the current page is aborted.	
Asp validators	RequiredFieldValidator: Enables you to require a user to enter a value in a form field. RangeValidator: Enables you to check whether a value falls between a certain minimum and maximum value. CompareValidator: Enables you to compare a value against another value or perform a data type check. RegularExpressionValidator: Enables you to compare a value against a regular expression. CustomValidator: Enables you to perform custom validation. ValidationSummary: Enables you to display a summary of all validation errors in a page.	
CompareValidator	ControlToValidate: The form element being validated. Text: The error indicator like #,* etc. ErrorMessage: Error message to be displayed if the validation fails. Type: The type of value being compared. Possible values are String, Integer, Double, Date, and Currency. Operator: The type of comparison to perform. Possible values are DataTypeCheck, Equal, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and NotEqual. ValueToCompare: The fixed value against which to compare. ControlToCompare: The ID of a control against which to compare.	
RegularExpressionValidator	ValidationExpression: The regular expression pattern to compare with the input value.	
CustomValidator	ClientValidationFunction: The name of a client-side function used to perform client-side validation. ServerValidate: This event is raised when the CustomValidator performs validation.	
	ValidationGroup	
Page Life cycle		http://msdn.microsoft.com/en-us/library/vstudio/ms178472(v=vs.100).aspx
Pre_Init	Creates the object of control and page(But it doesn't initializing the property)	
	We can create the control dynimacally can set the masterpage and Theme. Can create the datasource dynamically.	
Init	can read or set the property (Default property will be set) can create the control dynamically for content page.	
Init_complete	Can change the view state.last point to change the view state.	
Preload	After viewState,Data has been loaded for page and control at this point. After postback processing.	
Load	First for page and then for control. Can control PostBack event.	
Control PostBack Event		
LoadComplete		
Pre_render	Final changes to the page or its control. IT will called after all cotrol's postback. It will being called before viewstate save	The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.	
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control. All ASP.NET Web server controls have a Render method that writes out the control's markup to send to the browser.	
Unload	Raised for each control and then for the page.	
Master page	.master .master.aspx.cs <@master … >	
	<asp:ContentPlaceHolder > ContentPlaceHolderID <asp:Content> contentID ContentPlaceHolderID	
	MasterPageFile in <%@ Page %> directive	<configuration> <system.web> pages masterPageFile="~/Samplemaster.master" /> </system.web> </configuration>
	can change the master page from pre_init of the page life cycle	
To access the property from Master page into content page	1.Create the public property in master page 2.Add the @MasterType directive in page 3.Access the property in page as Master.<Property>	
.skin file	A default skin automatically applies to all controls of the same type when a theme is applied to a page. A named skin is a control skin with a SkinID property set.	
Theme		<% Page language = "C#" Theme = "MY Theme" %> <configuration> <system.web> <page theme="My theme" /> </system.web> <configuration>
Theme Vs StylesheetTheme	If you specify the Theme attribute in Page directive then setting in theme will take precedence and override any setting whichever u have specified in each control But if u have specify the StylesheetTheme attribute in Page directive then setting in each control will take precedence and override any setting whichever u have specified in Theme.	
EnableTheming	can disable the theme at the age level by specifiying this attribute at page level.	
Grid View		<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" BackColor="#DEBA84" BorderColor "#DEBA84" BorderStyle="None" BorderColor="#BorderStyle" BorderWidth="1px" CellPadding="3" CellSpacing="2" DataKeyNames="CustomerID" DataSourceID="SqlDataSource1" AllowPaging="True" AllowSorting="True">
Difference between DataGrid and GridView	1. DataGrid is introduced in asp.net 1.1 and is still supported today. GridView is introduced in asp.net 2.0. 2. Declarative datasource controls can be used with DataGrid only for data selection. Tasks like paging, sorting, deletes and updates must be done in code. The GridView control can achieve all of these using the declarative datasource controls. 3. GridView introduces new column types.	

Concepts	Description	Example
Data source	1.SqlDataSource - Use to work with SQL Server, OLE DB, ODBC, or Oracle databases 2.ObjectDataSource - Use to work business objects, that manages data 3.AccessDataSource - Use to work with Microsoft Access 4.XmlDataSource - Use to work with XML files 5.LinqDataSource - Enables us to use LINQ, to retrieve and modify data from a data object 6.EntityDataSource - Use to work with Entity Data Model	

SqlDataSource	Points to remember:
	1. "ConnectionString" property of the "SqlDataSource" control is used to determine the database it has to connect, to retrieve data
	2. "SelectCommand" property specifies the command that needs to be executed.
	3. DataSource control is associated, with the gridview control, using "DataSourceID" property of the GridView control.
		<asp:SqlDataSource ID="SqlDataSource1" runat="server"
		ConnectionString="<%$ ConnectionStrings:DBConnectionString %>"
		SelectCommand="SELECT * FROM [tblProducts]">
		</asp:SqlDataSource>

ObjectDataSource	Getting data from databse and assiging values to the property of the object and binding the data source object. Properties to remember : Select Method : name of the method which returns required busineess object. Type name : name of the class containg above method.	<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="GetAllProducts" TypeName="Demo.ProductDataAccessLayer"> </asp:ObjectDataSource>

XmlDataSource	Add the Xml file as source and add source file to XmlDatasource object.
	XmlDataSource control works with attributes, not child xml entities.
	We can of 3 ways to solve this issue
	1. Rewrite Countries.xml file, using attributes instead of xml nodes.
	2. Use an XSLT transformation file, to convert xml nodes to attributes.
	3. Load Countries.xml data into a dataset and then bind to the gridview control
		DataSet ds = new DataSet();
		ds.ReadXml(Server.MapPath("~/Data/Countries.xml"));
		GridView1.DataSource = ds;
		GridView1.DataBind();

	Property DataFormatString : read from here http://msdn.microsoft.com/en-gb/library/az4se3k1.aspx HeaderText: header of the column Visable:	<asp:BoundField DataField="EmployeeId" HeaderText="Employee Id" InsertVisible="False" ReadOnly="True" SortExpression="EmployeeId" DataFormatString="{0:D}", /> asp:BoundField DataField="Territory" HeaderText="Territory #" ItemStyle-HorizontalAlign="Left" HeaderStyle-HorizontalAlign="Left" />
onrowdatabound attribute	Will be called after the row will be binded with the grid each time	Needed if we want to do operation after each row binded.
ItemStyle-CssClass		
RowDeleted		
Optimistic Concurrency	Will not update any record if Optimistic Concurrency is checked.it will check whether data in Db is same as on page it is.	
DataKeyNames and ConvertEmptyStringToNull		
Affected rows and KeeptIn EditMode	Affected rows shows thr number of rows updated. KeepInEditMode keeps the field in Edit mode.	
<asp:TemplateField HeaderText="EmpID"> <ItemTemplate> </ItemTemplate> <EditItemTemplate> </EditItemTemplate> </asp:TemplateField>		
RowCommand	Is a event and it will be fired when any event will occur in gridview	
ondatabinding	Occurs when the server control binds to a data source	
ondatabound	Occurs after the server control binds to a data source.	
onpageindexchanging	Occurs when one of the pager buttons is clicked, but before the GridView control handles the paging operation.	
onpageindexchanged	Occurs when one of the pager buttons is clicked, but after the GridView control handles the paging operation.	
onselectedindexchanging	Occurs when a row's Select button is clicked, but before the GridView control handles the select operation	
onselectedindexchanged	Occurs when a row's Select button is clicked, but after the GridView control handles the select operation	
onRowCancelingEdit	Occurs when the Cancel button of a row in edit mode is clicked, but before the row exits edit mode.	
onrowediting	Occurs when a row's Edit button is clicked, but before the GridView control enters edit mode.	
onrowdeleted	Occurs when a row's Delete button is clicked, but after the GridView control deletes the row.	
onrowdeleting	Occurs when a row's Delete button is clicked, but before the GridView control deletes the row.	
onrowupdated	Occurs when a row's Update button is clicked, but after the GridView control updates the row.	
onrowupdating	Occurs when a row's Update button is clicked, but before the GridView control updates the row.	
onrowcreated :	While binding the grid,For each record in dataset,Gridview's row means GriedViewRow will be created.	
GridViewRowEventArgs :	Object of GridViewRowEventArgs will be created on onrowcreated event.It has ome property as object of type GridViewRow	
onrowdatabound:	Occurs when a data row is bound to data in a GridView control.	
onpageindexchanged :	Occurs when one of the pager buttons is clicked, but after the GridView control handles the paging operation.	
EventArgs		
onpageindexchanging :	Occurs when one of the pager buttons is clicked, but before the GridView control handles the paging operation	
GridViewPageEventArgs:	Object of GridViewPageEventArgs will be created on onpageindexchanging event.It has one property NewPageIndex.	
onsorting :	Occurs when the hyperlink to sort a column is clicked, but before the GridView control handles the sort operation.	
GridViewSortEventArgs :	Object of GridViewSortEventArgs will be created on onsorting event.It has two property SortExpression(type : String) and SortDirection (Type : SortDirection) How to use SortDirection ??	
onsorted	Occurs when the hyperlink to sort a column is clicked, but after the GridView control handles the sort operation	
GridViewCommandEventArgs		
GridViewEditEventArgs	Object of GridViewEditEventArgs will be passed OnRowEditing event. It has three member.1.RowIndex 2.keys 3.Value	
GridViewPageEventArgs		
GridViewDeletedEventArgs	Object of GridViewDeletedEventArgs will be passed onrowdeleted event. It has 5 properties.1.AffectedRows 2.Keys 3.Values, 4.Exception 5. ExceptionHandled	
GridViewDeleteEventArgs	Object of GridViewDeleteEventArgs will be passed onrowdeleting event. It has 4 properties.1.RowIndex 2.Keys 3.NewValues,4.OldValues	
GridViewUpdatedEventArgs	Object of GridViewUpdateEventArgs will be passed OnRowUpdating event. It has 4 properties.1.RowIndex 2.Keys 3.NewValues,4.OldValues	
GridViewUpdateEventArgs	Object of GridViewUpdatedEventArgs will be passed OnRowUpdated event. It has 7 properties.1.AffectedRows 2.Keys 3.NewValues,4.OldValues 5. Exception 6.ExceptionHandled 7.KeepInEditMode	
GridViewCancelEditEventArgs	Object of GridViewCancelEditEventArgs will be passed onrowcancelingedit event. It has only one property. 1. RowIndex	
GridViewSelectEventArgs	Object of GridViewSelectEventArgs will be passed onselectedindexchanging event. It has one property.1.NewSelectedIndex	
EventArgs	Object of EventArgs will be passed onselectedindexchanged event. It has one property.1.NewSelectedIndex	
GriedViewRow		

Concepts	Description	Example				
NamingContainer	You can use the NamingContainer property of a naming container's child control to get a reference to its parent container.					
<emptyDatTemple> **</emptyDatTemple>** **EmptydataText proerty of GridView**	Whern there is no record					
REPEATER						
onitemcommand :	Occurs when a button is clicked in the Repeater control					
RepeaterCommandEventArgs :	Object of RepeaterCommandEventArgs will be created on onsorting event.It has two property CommandSource(type : object) and Item (Type : RepeaterItem)					
onitemcreated:	Occurs when an item is created in the Repeater control. RepeaterItemEventArgs: Object of RepeaterCommandEventArgs will be created on onsorting event.It has one property Item (Type : RepeaterItem)					
onitemdatabound :	Occurs after an item in the Repeater control is data-bound but before it is rendered on the page.					
RepeaterItemEventArgs:	Templates availabale in ASP.Net ItemTemplate AlternativeItemTemplate FooterTemplate Headertemplate SeparatorTemplate					
List View :						
SelectedIndexChanging Event :	Occurs when an item's Select button is clicked, but before the ListView control handles the select operation.					
onselectedindexchanging:	Occurs when an item's Select button is clicked, but before the ListView control handles the select operation.					
ListViewSelectEventArgs :	Property : NewSelectedIndex (type : int)					
Application Caching :	Can store any object in memory It is a process of storing any object in cache object. Page.Cache property acually uses an application wide cache. Cache can be shared between session and requets					
Parameters of insertion of cache	Key : Value: Dependancy: AbsoluteExiration: slidingExpiration: priority: onRemoveCallBack:					
Dependancy:	This will identify the file or another cache object.when this file or cache object will be changed,this cache object will be removed.					
AbsoluteExiration:	it specify at what time cache will be expired.doesn't matter if we have used the object recently or not. If we don't want to use this .we can set System.Web.Caching.Cache. NoAbsoluteExipration.					
slidingExpiration:	after specifed time,if object is not used then object will be removed from the cache. System.Web.Caching.Cache.NoSlidingExpiration					
Page Output Caching						
VaryByParam	<%@ outputcache duration = "15" VaryByParam="none" >					
VaryByControl						
VaryByCustom						
VaryByHeader						
Partial page caching						
FormView	The FormView control lets you work with a single record from a data source, similar to the DetailsView control. The difference between the FormView and the DetailsView controls is that the DetailsView control uses a tabular layout where each field of the record is displayed as a row of its own. In contrast, the FormView control does not specify a pre-defined layout for displaying the record.					
	The formView control is completely template driven					
FormView vs DetailView	•The DetailsView control is easier to work with. •The FormView control provides more control over the layout. •The FormView control uses only the templates with databinding expressions to display data. The DetailsView control uses <BoundField> elements or <TemplateField> elements. •The FormView control renders all fields in a single table row whereas the DetailsView control displays each field as a table row. •DetailsView has a built-in tabular rendering, whereas FormView requires a user-defined template for its rendering					
Eval and Bind function,CommandName="Sort" CommandArgument="ID"	Eval is one-way, read only databinding. Bind is two-way, read/write databinding.					
Difference between Gried View vs datagrid vs repeater vs ListView						
ASP.net Ajax						
Page directive	The @Page directive enables you to specify attributes and values for an Asp.Net Page to be used when the page is parsed and compiled. Complier will use the information while compilling.	b. Language: This attribute tells the compiler about the language being used in the code-behind. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET. c. AutoEventWireup: For every page there is an automatic way to bind the events to methods in the same .aspx file or in code behind. The default value is true. d. CodeFile: Specifies the code-behid file with which the page is associated. e. Title: To set the page title other than what is specified in the master page. f. Culture: Specifies the culture setting of the page. If you set to auto, enables the page to automatically detect the culture required for the page. k. MasterPageFile: Specify the location of the MasterPage file to be used with the current Asp.Net page. l. EnableViewState: Indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true. n. Inherits: Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.				

AutoEventWireup	When AutoEventWireup is true, handlers are automatically bound to events at run time based on their name and signature. For each event, ASP.NET searches for a method that is named according to the pattern Page_eventname, such as Page_Load or Page_Init.
	When AutoEventWireup is false, you must explicitly bind event handlers to events, as shown in the preceding example. In that case, the method names do not have to follow a pattern.
		public partial class AutoEventWireupExample : System.Web.UI.Page
		{
		  protected void Page_Load(object sender, System.EventArgs e)
		  {
		    Response.Write("Executing Page_Load");
		  }
		  override protected void OnInit(EventArgs e)
		  {
		    this.Load += new System.EventHandler(this.Page_Load);
		  }
		}				

@Master	The @Master directive is quite similar to the @Page directive. The @Master directive belongs to Master Pages that is .master files. The master page will be used in conjunction of any number of content pages. So the content pages can the inherits the attributes of the master page. Even though, both @Page and @Master page directives are similar, the @Master directive has only fewer attributes	a. Language: b. AutoEventWireup: c. CodeFile: d. Title: e. MasterPageFile: f. EnableViewState: g. Inherits:				
@Control	The @Control directive helps us to define the properties to be inherited by the user control.	a. Language: b. AutoEventWireup: c. CodeFile: d. EnableViewState: e. Inherits: f. Debug: g. Src:				
What is the difference between disable control and read only control				**GridView**	**ListView**	**DataList**

Concepts	Description	Example				
GridView	Repeater		Repeater	No	Yes	Yes
Table layout by default	Uses templates	**Flow layout**	Yes	Yes	No	No
Has select/edit/delete commands	Must be added manually	**Table layout**	No	Yes	Yes	Yes
Built-in pager support	Must be added manually	**Style properties**	No	No	No	Yes
Column sorting	Must be added manually	**Column layout**	No	Yes	Yes	No
		Paging	No	Yes	Yes	No
		Sorting	No	Yes	Yes	No
		Edit/Delete	No	No	Yes	No
		Insert	No	No	Yes	Yes
		Grouping	No			
ASP.NET 4.0 Features						
Web.config File Refactoring	The Web.config file that contains configuration information for a Web application has grown considerably over the past few releases of the .NET Framework as new features have been added. In the .NET Framework 4, the major configuration elements have been moved to the machine.config file, and applications now inherit these settings.	How its possible that if we move application specific setting to machine.config file? What are the stuff that we can configure in configaration file.	http://msdn.microsoft.com/en-us/library/vstudio/s57a598e(v=vs.100).aspx			
Machine.config vs web.config	The settings of Machine.config file are applied to the whole asp.net applications on your server whereas the settings made in the Web.config file are applied to that particular web application only. web application will immediately load the changes but in case of machine.config you will have to restart the application.	Web.config file will override the machine.config file's setting.				
Extensible Output Caching	Framework is extended to have cache providers so that it can cache the different version of web pages.	Need to check in details.Not so important because we don't have so static pages that we can save in cache.				
Auto-Start Web Applications	Some Web applications must load large amounts of data or must perform expensive initialization processing before serving the first request. In earlier versions of ASP.NET, for these situations you had to devise custom approaches to "wake up" an ASP.NET application and then run initialization code during the Application_Load method in the Global.asax file. To address this scenario, a new auto-start feature is available when ASP.NET 4 runs on IIS 7.5 on Windows Server 2008 R2. The feature provides a controlled approach for starting up an application pool, initializing an ASP.NET application, and then accepting HTTP requests. It lets you perform expensive application initialization prior to processing the first HTTP request.	How to implement the feature				
Permanently Redirecting a Page						
Session State Compression	You can set this option using the new compressionEnabled attribute of the sessionState element in the configuration file. When the compressionEnabled configuration option is set to true, ASP.NET compresses (and decompresses) serialized session state by using the .NET Framework GZipStream class.	Are we really pass the entire session object to client or we pass only session ID ?				
Expanding the Range of Allowable URLs	you have the option to increase (or decrease) this limit as appropriate for your applications, using two new attributes of the httpRuntime configuration element.	<httpRuntime maxRequestPathLength="260" maxQueryStringLength="2048" /> What was the limit earlier ? And what is the limit now.				
	ASP.NET 4 also enables you to configure the characters that are used by the URL character check. When ASP.NET finds an invalid character in the path portion of a URL, it rejects the request and issues an HTTP 400 (Bad request) status code. In previous versions of ASP.NET, the URL character checks were limited to a fixed set of characters. In ASP.NET 4, you can customize the set of valid characters using the new requestPathInvalidChars attribute of the httpRuntime configuration element, as shown in the following example:	<httpRuntime requestPathInvalidChars="&lt;,&gt;,*,%,&amp;,:,\,?" />				
ExtensibleRequest validation	ASP.NET request validation searches incoming HTTP request data for strings that are typically used in cross-site scripting (XSS) attacks. If potential XSS strings are found, request validation flags the suspect string and returns an error. You can customize the built in logic and implement your own logic to validagte the requested data.					
Extensible HTML, URL, and HTTP Header Encoding	?? What is Encoding and HTTP Encoding ? HTTP Attribute Encoding ? URL Encoding ?					
Setting Meta Tags with the Page.MetaKeywords and Page.MetaDescription Properties	Two properties have been added to the Page class: MetaKeywords and MetaDescription. These two properties represent corresponding meta tags in the HTML rendered for a page	<head id="Head1" runat="server"> <title>Untitled Page</title> <meta name="keywords" content="keyword1, keyword2' /> <meta name="description" content="Description of my page" /> </head>				
Enabling View State for Individual Controls	A new property has been added to the Control class: ViewStateMode. You can use this property to disable view state for all controls on a page except those for which you explicitly enable view state.	The ViewStateMode property of a page or a control has an effect only if the EnableViewState property is set to true. If the EnableViewState property is set to false, view state will be turned off even if the ViewStateMode property is set to Enabled. To disable view state for a page and to enable it for a specific control on the page, set the EnableViewState property of the page and the control to true, set the ViewStateMode property of the page to Disabled, and set the ViewStateMode property of the control to Enabled.				
Routing in ASP.NET 4						
Client ID						
Persisting Row Selection in Data Controls	The GridView and ListView controls enable users to select a row. In previous versions of ASP.NET, row selection was based on the row index on the page. For example, if you select the third item on page 1 and then move to page 2, the third item on page 2 is selected. In most cases, is more desirable not to select any rows on page 2. ASP.NET 4 supports Persisted Selection, a new feature that was initially supported only in Dynamic Data projects in the .NET Framework 3.5 SP1. When this feature is enabled, the selected item is based on the row data key. This means that if you select the third row on page 1 and move to page 2, nothing is selected on page 2. When you move back to page 1, the third row is still selected. This is a much more natural behavior than the behavior in earlier versions of ASP.NET. Persisted selection is now supported for the GridView and ListView controls in all projects. You can enable this feature in the GridView control, for example, by setting the EnablePersistedSelection property, as shown in the following example:	<asp:GridView id="GridView2" runat="server" PersistedSelection="true"> </asp:GridView>				
FormView Control Enhancements	The FormView control supports RenderOuterTable, a property in ASP.NET 4. When this property is set to false, as show in the following example, the table tags are not rendered. This makes it easier to apply CSS style to the contents of the control.	<asp:FormView ID="FormView1" runat="server" RenderTable="false">				
ListView Control Enhancements						
Dynamic Control	?? Reading is pending					
ASP.NET 4.5 Features						
New Request Validation features in Asp.Net 4.5	Asp.Net 4.5 added a new property to every server control called ValidateRequestMode. Set it to "disabled"	http://www.codeproject.com/Articles/632212/Asp-Net-features-Part				
Bundling						
Server.Execute	Server.Transfer and Server.Execute are similar in many ways. 1. The URL in the browser remains the first page URL. 2. Server.Transfer and Server.Execute can only be used to navigate to sites/pages on the same web server. Trying to navigate to sites/pages on a different web server, causes runtime exception. 3. Server.Transfer and Server.Execute preserves the Form Variables from the original request.					
	The major difference between Server.Transfer and Server.Execute is that, Server.Transfer terminates the execution of the current page and starts the execution of the new page, where as Server.Execute process the second Web form without leaving the first Web form. After completing the execution of the first webform, the control returns to the second webform and return both the pages.					
Cross page posting:	Cross page posting allows to post one page to another page. By default, when you click a button, the webform posts to itself. If you want to post to another webform on a button click, set the PostBackUrl of the button, to the page that you want to post to.					

Concepts	Description	Example	

Cookies:	Cookies can also be used to send data from one webform to another. In general, web sites use cookies to store user preferences or other information that is client-specific. Cookies store small amounts of information on the client's machine.
	Cookies can be broadly classified into 2 types
	1. Persistent cookies - Remain on the client computer, even after the browser is closed. You can configure how long the cookies remain using the expires property of the HttpCookie object.
	2. Non-Persistent cookies - If you don't set the Expires property, then the cookie is called as a Non-Persistent cookie. Non-Persistent cookies only remain in memory until the browser is closed
		// Create the cookie object
		HttpCookie cookie = new HttpCookie("UserDetails");
		cookie["Name"] = txtName.Text;
		cookie["Email"] = txtEmail.Text;
		// Cookie will be persisted for 30 days
		cookie.Expires = DateTime.Now.AddDays(30);
		// Add the cookie to the client machine
		Response.Cookies.Add(cookie);
		
		//Retrive coockies
		HttpCookie cookie = Request.Cookies["UserDetails"];
		if (cookie != null)
		{
		lblName.Text = cookie["Name"];
		lblEmail.Text = cookie["Email"];
		}	

	Generally cookies are stored on local system by coockies by default.but if user disables coockies,site wont work as expected. To enable cookieless sessions, set cookieless="true" in web.config as shown below. <sessionState mode="InProc" cookieless="true"></sessionState> session-id is now part of the URL		
session state mode as Inporc :	When the session state mode is set to InProc, the session state variables are stored on the web server memory inside the asp.net worker process. This is the default session state mode.		
Web Garden	Web application deployed on a server with multiple processors		
Web Farm	Web application deployed on multiple server		
session state mode as StateServer :	The session state variables are stored in a process, called as asp.net state service. This process is different from the asp.net worker process. The asp.net state service can be present on a web server or a dedicated machine.	Example: <sessionState mode="StateServer" stateConnectionString="tcpip=localhost: 42424" timeout="20"></sessionState>	ref :http://csharp-video-tutorials.blogspot.in/2012/12/stateserver-aspnet-session-state-mode.html
session state mode as SQLServer :	the session state variables are stored in a SQLServer database.		
sending mail using asp.net	http://csharp-video-tutorials.blogspot.in/2012/12/sending-emails-using-aspnet-part-77.html		
What is the difference between VaryByParam and VaryByControl?	If you want to cache multiple responses of a user control, based on a query string or a form "POST" parameter, then use VaryByParam. On the other hand, if you want to cache multiple responses of a user control, based on a control value then use "VaryByControl".	Ref :http://csharp-video-tutorials.blogspot.in/2013/02/caching-multiple-versions-of-user_6.html	
Caching application data	1. Cache["ProductsData"] = ds; DataSet ds = (DataSet)Cache["ProductsData"]; 2. Using "Cache" object's Insert() method : Insert() method has got 5 overloaded versions 3. Using "Cache" object's Add() method : Cache.Add("ProductsData", ds, null, System.Web.Caching.Cache.NoAbsoluteExpiration, System.Web.Caching.Cache.NoSlidingExpiration, System.Web.Caching.CacheItemPriority.Default, null);		
CacheItemPriority:	Sliding expiration and absolute expiration can be used to control, how long the item is cached, but please note, if the web server is running low on memory, and if it requires memory, it may remove cached items that may not have expired. However, the order in which the items are removed is determined by the cached item's priority. Cache item's priority can be specified using CacheItemPriority enum.	CacheItemPriority enum values: CacheItemPriority.Low CacheItemPriority.BelowNormal CacheItemPriority.Normal CacheItemPriority.Default CacheItemPriority.AboveNormal CacheItemPriority.High CacheItemPriority.NotRemovable	
Cache dependency :	Cache.Insert("CountriesData", ds, new CacheDependency(Server.MapPath("~/Data/Countries.xml")), DateTime.Now.AddSeconds(20), System.Web.Caching.Cache.NoSlidingExpiration);		
CacheItemRemovedCallback	Need to pass the object of delegate CacheItemRemovedCallback as a parameter and method associated with the delegate will be called on removal of chache. public delegate void CCacheItemRemovedCallback(string key, object value, CacheItemRemovedReason reason);		
to display an icon for website on browser tab	add this piece of code in header block. <link rel="shortcut icon" href="~/favicon.ico" type="image/x-icon"/>		
Sever.MapPath			
How IIS Serve request		https://www.codeproject.com/articles/121096/web-server-and-asp-net-application-life-cycle-in-d	
Aspnet_regiis.dll			
ApplicationPool			
Application manager			
HttpApplication			
When we run our ASP.NET Web Application from visual studio IDE, VS Integrated ASP.NET Engine is responsible to execute all kind of asp.net requests and responses. The process name is “WebDev.WebServer.Exe” which actually take care of all request and response of an web application which is running from Visual Studio IDE			
Worker Process (w3wp.exe)		https://www.codeproject.com/Articles/73728/ASP-NET-Application-and-Page-Life-Cycle	
One confusion: Which process run which code		https://www.codeproject.com/articles/32475/asp-net-http-modules	
HttpRuntime		https://msdn.microsoft.com/en-us/library/bb470252.aspx	
Http handlers			
Http Modules			
Web hosting			
How IIS Serve respond to request			

Revisit HttpEvent	https://angular.io/guide/http#httpevents		

The clone() method's hash argument allows you to mutate specific properties of the request while copying the others	// clone request and replace 'http://' with 'https://' at the same time
	const secureReq = req.clone({
	  url: req.url.replace('http://', 'https://')
	});
	// send the cloned, "secure" request to the next handler.
	return next.handle(secureReq);		

RxJs Operator in Angular	*debounceTime(500)*		
	distinctUntilChanged		
	switchMap, Map Operator		
	pipe		
	catchError		
	retry()		
	tap, catcherror, throwError ,map		

Reading the full response	Tell HttpClient that you want the full response with the observe option.
	HttpClient.get() returns an Observable of typed HttpResponse rather than just the JSON data.
		getConfigResponse(): Observable<HttpResponse<Config>> {
		  return this.http.get<Config>(
		    this.configUrl, { **observe: 'response'** });
		}	

Error handling	You could handle in the component by adding a second callback to the .subscribe()	.subscribe(
		      (data: Config) => this.config = { ...data }, // success path
		      error => this.error = error // error path
		    );	

	Two types of errors can occur. The server backend might reject the request, returning an HTTP response with a status code such as 404 or 500. These are error responses. Or something could go wrong on the client-side such as a network error that prevents the request from completing successfully or an exception thrown in an RxJS operator. These errors produce JavaScript ErrorEvent objects.		
	You must call subscribe() or nothing happens. Just calling HeroesService.deleteHero() does not initiate the DELETE(Any) request		
Update headers	You can't directly modify the existing headers within the previous options object because instances of the HttpHeaders class are immutable. Use the set() method instead. It returns a clone of the current instance with the new changes applied. Here's how you might update the authorization header (after the old token expired) before making the next request.	httpOptions.headers = httpOptions.headers.set('Authorization', 'my-new-auth-token');	
Intercepting requests and responses	With interception, you declare interceptors that inspect and transform HTTP requests from your application to the server. The same interceptors may also inspect and transform the server's responses on their way back to the application. Multiple interceptors form a forward-and-backward chain of request/response handlers.		

Write an interceptor	To implement an interceptor, declare a class that implements the intercept() method of the HttpInterceptor interface.	import { Injectable } from '@angular/core';
		import {
		  HttpEvent, HttpInterceptor, HttpHandler, HttpRequest
		} from '@angular/common/http';
		
		import { Observable } from 'rxjs';
		
		/** Pass untouched request through to the next request handler. */
		@Injectable()
		export class NoopInterceptor implements HttpInterceptor {
		
		  intercept(req: HttpRequest<any>, next: HttpHandler):
		    Observable<HttpEvent<any>> {
		    return next.handle(req);
		  }
		}	

Provide the interceptor		export const httpInterceptorProviders = [
		  { provide: HTTP_INTERCEPTORS, useClass: NoopInterceptor, multi: true },
		];
		
		providers: [
		  httpInterceptorProviders
		],	

Interceptor order	Angular applies interceptors in the order that you provide them. If you provide interceptors A, then B, then C, requests will flow in A->B->C and responses will flow out C->B->A. You cannot change the order or remove interceptors later. If you need to enable and disable an interceptor dynamically, you'll have to build that capability into the interceptor itself.		

Listening to progress events	Sometimes applications transfer large amounts of data and those transfers can take a long time. File uploads are a typical example. Give the users a better experience by providing feedback on the progress of such transfers.
	
	To make a request with progress events enabled, you can create an instance of HttpRequest with the reportProgress option set true to enable tracking of progress events.
	
	app/uploader/uploader.service.ts (upload request)
		const req = new HttpRequest('POST', '/upload/file', file, {
		  reportProgress: true
		});	

	Rx will help to use HttpClient mode efficiently...		
<base href>	Most routing applications should add a <base> element to the index.html as the first child in the <head> tag to tell the router how to compose navigation URLs. If the app folder is the application root, as it is for the sample application, set the href value exactly as shown here. The HTML <base> tag is used to specify a base URI, or URL, for relative links. For example, you can set the base URL once at the top of your page, then all subsequent relative links will use that URL as a starting point.	<base href="/">	
Router feature		import { RouterModule, Routes Router, NavigationStart, NavigationCancel, NavigationError, NavigationEnd CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router CanDeactivate Router, ActivatedRoute } from '@angular/router'	
		Router: Displays the application component for the active URL. Manages navigation from one component to the next. RouterModule: A separate NgModule that provides the necessary service providers and directives for navigating through application views. Routes: Defines an array of Routes, each mapping a URL path to a component. Route: Defines how the router should navigate to a component based on a URL pattern. Most routes consist of a path and a component type. RouterOutlet: The directive (<router-outlet>) that marks where the router displays a view.	
<Explore more: ActivatedRoute and RouterState>		RouterLink : The directive for binding a clickable HTML element to a route. Clicking an element with a routerLink directive that is bound to a string or a link parameters array triggers a navigation. RouterLinkActive: The directive for adding/removing classes from an HTML element when an associated routerLink becomes active/inactive respectively. ActivatedRoute: A service that is provided to each route component that contains route specific information such as route parameters, static data, resolve data, global query params, and the global fragment. RouterState: The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree.	
		Link parameters array: An array that the router interprets as a routing instruction. You can bind that array to a RouterLink or pass the array as an argument to the Router.navigate method. Routing component: An Angular component with a RouterOutlet that displays views based on router navigations.	

Router class in RouterModule	Router service basically helps to navigate to other route.	Provides below usefull behaviours:
		resetConfig(config: Routes) //Resets the configuration used for navigation and generating links.
		createUrlTree()
		navigateByUrl(url: string | UrlTree, extras?: NavigationExtras) //Navigate based on the provided url. This navigation is always absolute.
		navigate(commands: any[], extras?: NavigationExtras)//Navigate based on the provided array of commands and a starting point.
		serializeUrl(url: UrlTree): string; //Serializes a `UrlTree` into a string
		parseUrl(url: string): UrlTree; //Parses a string into a `UrlTree
		isActive(url: string | UrlTree, exact: boolean) //Returns whether the url is activated	

RouterLinkActive		The template expression to the right of the equals (=) contains a space-delimited string of CSS classes that the Router will add when this link is active (and remove when the link is inactive). You set the RouterLinkActive directive to a string of classes such as [routerLinkActive]="'active fluffy'" or bind it to a component property that returns such a string.	

Route Class in RouterModule	Available option for route configuration.	path?: string;
		    pathMatch?: string;
		    matcher?: UrlMatcher;
		    component?: Type<any>;
		    redirectTo?: string;
		    outlet?: string; ==> ??
		    canActivate?: any[];
		    canActivateChild?: any[];
		    canDeactivate?: any[];
		    canLoad?: any[];
		    data?: Data;
		    resolve?: ResolveData;
		    children?: Routes;
		    loadChildren?: LoadChildren;
		    runGuardsAndResolvers?: RunGuardsAndResolvers;	

Singleton implementation in Angular	Please refer the link for implementing service as singleton	https://angular.io/guide/singleton-services#forroot	
	A routed Angular application has one singleton instance of the Router service. When the browser's URL changes, that router looks for a corresponding Route from which it can determine the component to display.	https://angular.io/guide/singleton-services#forroot	
	The appRoutes array of routes describes how to navigate. Pass it to the RouterModule.forRoot method in the module imports to configure the router.		
leading slashes			
Passing Token as parameter		{ path: 'hero/:id', component: HeroDetailComponent },	
	The data property in this route is a place to store arbitrary data associated with this specific route. The data property is accessible within each activated route. Use it to store items such as page titles, breadcrumb text, and other read-only, static data. You'll use the resolve guard to retrieve dynamic data later in the guide.	{ path: 'heroes', component: HeroListComponent, data: { title: 'Heroes List' } }	
	The empty path in the fourth route represents the default path for the application, the place to go when the path in the URL is empty, as it typically is at the start. This default route redirects to the route for the /heroes URL and, therefore, will display the HeroesListComponent.	{ path: '', redirectTo: '/heroes', pathMatch: 'full' },	

	Technically, pathMatch = 'full' results in a route hit when the remaining, unmatched segments of the URL match ''. In this example, the redirect is in a top level route so the remaining URL and the entire URL are the same thing.		
	The other possible pathMatch value is 'prefix' which tells the router to match the redirect route when the remaining URL begins with the redirect route's prefix path		
	The ** path in the last route is a wildcard. The router will select this route if the requested URL doesn't match any paths for routes defined earlier in the configuration. This is useful for displaying a "404 - Not Found" page or redirecting to another route.	{ path: '**', component: PageNotFoundComponent }	
	The router uses a first-match wins strategy when matching routes, so more specific routes should be placed above less specific routes.		
	You set the enableTracing: true option in the object passed as the second argument to the RouterModule.forRoot() method.	RouterModule.forRoot(appRoutes, { enableTracing: true })	
Router outlet	The RouterOutlet is a directive from the router library that is used like a component. It acts as a placeholder that marks the spot in the template where the router should display the components for that outlet.	<router-outlet></router-outlet>	
Router links	routerLink to redirect to other component in html	<a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>	
Router state	After the end of each successful navigation lifecycle, the router builds a tree of ActivatedRoute objects that make up the current state of the router. You can access the current RouterState from anywhere in the application using the Router service and the routerState property. Each ActivatedRoute in the RouterState provides methods to traverse up and down the route tree to get information from parent, child and sibling routes.		
ActivatedRoute	ActivatedRoute is a service provides services like route path and data. Each ActivatedRoute in the RouterState provides methods to traverse up and down the route tree to get information from parent, child and sibling routes.		
	snapshot is important property data: Example of resolved data - this.route.snapshot.data['contact']; To get resolved data from snalshot property.	url => ?? An Observable of the route path(s), represented as an array of strings for each part of the route path. data ==> Resolved data or values(static data) mentioned in that route specified in Route configuration of data An Observable that contains the data object provided for the route. Also contains any resolved values from the resolve guard. paramMap An Observable that contains a map of the required and optional parameters specific to the route. The map supports retrieving single and multiple values from the same parameter. queryParamMap An Observable that contains a map of the query parameters available to all routes. The map supports retrieving single and multiple values from the query parameter. fragment => ?? An Observable of the URL fragment available to all routes. outlet The name of the RouterOutlet used to render the route. For an unnamed outlet, the outlet name is primary.	https://vsavkin.com/angular-router-understanding-router-state-7b5b95a12eab https://blog.angularindepth.com/angular-router-series-secondary-outlets-primer-139206595e2
		routeConfig The route configuration used for the route that contains the origin path. parent The route's parent ActivatedRoute when this route is a child route. firstChild Contains the first ActivatedRoute in the list of this route's child routes. children Contains all the child routes activated under the current route.	https://www.techiediaries.com/angular-router-multiple-outlets/
	snapshot is type of ActivatedRouteSnapshot	https://vsavkin.com/angular-router-understanding-router-state-7b5b95a12eab	
ActivatedRoute Vs ActivatedRouteSnapshot (Component Reuse)			
RouterState and RouterStateSnapshot, tree of activated route and tree of activated routesnapshot			
Link parameters array Or Passing Required Parameter	Link parameter array contains first element as string (Path) and other elements as required parameter.	this.router.navigate(['../Hero', { id: crisisId, foo: 'foo' }]); Or <a routerLink="/crisis-center" routerLink="['../Hero', { id: crisisId, foo: 'foo' }]">Crisis Center</a>	
Optional Parameters	Query parameters allow you to pass optional parameters to a route. this.router.navigate(['/product-list'], { queryParams: { page: pageNum } }); queryParams property of NavigationExtras type is used to pass the optional parameters.		

NavigationExtras	Property of the NavigationExtras type:
	relativeTo?: ActivatedRoute | null;
	queryParams?: Params | null;
	fragment?: string;
	preserveQueryParams?: boolean;
	queryParamsHandling?: QueryParamsHandling | null;
	preserveFragment?: boolean;
	skipLocationChange?: boolean;
	replaceUrl?: boolean;		

queryParamsHandling	You can also preserve query parameters and fragments across navigations without having to provide them again when navigating.	let navigationExtras: NavigationExtras = {
		  queryParamsHandling: 'preserve',
		  preserveFragment: true
		};	

How relative path works in Angular router module.			
Router events		NavigationStart An event triggered when navigation starts. RouteConfigLoadStart An event triggered before the Router lazy loads a route configuration. RouteConfigLoadEnd An event triggered after a route has been lazy loaded. RoutesRecognized An event triggered when the Router parses the URL and the routes are recognized. GuardsCheckStart An event triggered when the Router begins the Guards phase of routing. ChildActivationStart An event triggered when the Router begins activating a route's children.	
<need to further explore by implementing those events>		ActivationStart An event triggered when the Router begins activating a route. GuardsCheckEnd An event triggered when the Router finishes the Guards phase of routing successfully. ResolveStart An event triggered when the Router begins the Resolve phase of routing. ResolveEnd An event triggered when the Router finishes the Resolve phase of routing successfuly. ChildActivationEnd An event triggered when the Router finishes activating a route's children. ActivationEnd An event triggered when the Router finishes activating a route. NavigationEnd An event triggered when navigation ends successfully.	
		NavigationCancel An event triggered when navigation is canceled. This is due to a Route Guard returning false during navigation. NavigationError An event triggered when navigation fails due to an unexpected error. Scroll An event that represents a scrolling event.	
forChild()	Only call RouterModule.forRoot in the root AppRoutingModule (or the AppModule if that's where you register top level application routes). In any other module, you must call the RouterModule.forChild method to register additional routes. -Why So	RouterModule.forChild(heroesRoutes)	
<REQUIRED OR OPTIONAL?>	prefer a required route parameter when the value is mandatory (for example, if necessary to distinguish one route path from another); prefer an optional parameter when the value is optional, complex, and/or multivariate	localhost:4200/heroes;id=15;foo=foo (Optional) The optional route parameters are not separated by "?" and "&" as they would be in the URL query string. They are separated by semicolons ";" This is matrix URL notation—something you may not have seen before.	
Observable paramMap and component reuse	snapshot vs paramMap and component instance reuse ParamMap is an Observable of type ParamMap. ParamMap provides api to access router parameter (Parameter Token). where Snapshot provides activatedRoute data of any given time. So whenever token parameter values get change, Router module won't create the new instance of the component but the componenet instance will be reused but will update the paramMap value (Emit the value of Parammap) so if in code, there is a subscriber on Parammap then it will be called. and if we have used snapshot then view wont be updated based on the new parameter value.	https://angular.io/guide/router#observable-parammap-and-component-reuse	

decode code	paramMap + switchMap + pipe	this.heroes$ = this.route.paramMap.pipe(
		    switchMap(params => {
		      // (+) before `params.get()` turns the string into a number
		      this.selectedId = +params.get('id');
		      return this.service.getHeroes();
		    })
		  );	

Relative navigation	To navigate a relative path with the Router.navigate method, you must supply the ActivatedRoute to give the router knowledge of where you are in the current route tree. After the link parameters array, add an object with a relativeTo property set to the ActivatedRoute. The router then calculates the target URL based on the active route's location.	The router supports directory-like syntax in a link parameters list to help guide route name lookup: ./ or no leading slash is relative to the current level. ../ to go up one level in the route path.	
Route Parameter	Routes parameters are only dealt with parameters specific to the route where query parameters are available to all routes.	this.router.navigate(['../', { id: crisisId, foo: 'foo' }], { relativeTo: this.route });	

Lazy load the module		{
		  path: 'admin',
		  loadChildren: './admin/admin.module#AdminModule',
		},	

	Webpack is a bundling tool		
@NgModule	@NgModule decorator identifies AppModule as an NgModule class. @NgModule takes a metadata object that tells Angular how to compile and launch the application.		
it has following properties	declarations: this application's lone component. The set of components, directives, and pipes that belong to this module. The set of selectors that are available to a template include those declared here, and those that are exported from imported NgModules. Declarables must belong to exactly one module. Be careful not to declare a class that is imported from another module.		
	imports: The set of NgModules whose exported are available to templates in this module. A template can use exported declarables from any imported module, including those from modules that are imported indirectly and re-exported. For example, `ModuleA` imports `ModuleB`, and also exports it, which makes the declarables from `ModuleB` available wherever `ModuleA` is imported.		
	providers: Dependencies whose providers are listed here become available for injection into any component, directive, pipe or service that is a child of this injector. The NgModule used for bootstrapping uses the root injector, and can provide dependencies to any part of the app. A lazy-loaded module has its own injector, typically a child of the app root injector. Lazy-loaded services are scoped to the lazy-loaded module's injector. If a lazy-loaded module also provides the `UserService`, any component created within that module's context (such as by router navigation) gets the local instance of the service, not the instance in the root injector. Components in external modules continue to receive the instance provided by their injectors.		
	bootstrap: The set of components that are bootstrapped when this module is bootstrapped. The components listed here are automatically added to `entryComponents`.		
	entryComponents: The set of components to compile when this NgModule is defined, so that they can be dynamically loaded into the view. *For each component listed here, Angular creates a `ComponentFactory` and stores it in the `ComponentFactoryResolver`.* *Angular automatically adds components in the module's bootstrap and route definitions into the `entryComponents` list.* *Use this option to add components that are bootstrapped using one of the imperative techniques, such as `ViewContainerRef.createComponent()`.*	??	
	exports: The set of components, directives, and pipes declared in this NgModule that can be used in the template of any component that is part of an NgModule that imports this NgModule. Exported declarations are the module's public API. A declarable belongs to one and only one NgModule. A module can list another module among its exports, in which case all of that module's public declaration are exported. @usageNotes * Declarations are private by default. * If this ModuleA does not export UserComponent, then only the components within this ModuleA can use UserComponent. * ModuleA can import ModuleB and also export it, making exports from ModuleB available to an NgModule that imports ModuleA.		
	schemas: *The set of schemas that declare elements to be allowed in the NgModule.* *Elements and properties that are neither Angular components nor directives must be declared in a schema.* *Allowed value are `NO_ERRORS_SCHEMA` and `CUSTOM_ELEMENTS_SCHEMA`.* *@security When using one of `NO_ERRORS_SCHEMA` or `CUSTOM_ELEMENTS_SCHEMA` you must ensure that allowed elements and properties securely escape inputs.*	??	
	The class we have created provides a service. The @Injectable() decorator marks it as a service that can be injected, but Angular can't actually inject it anywhere until you configure an Angular dependency injector with a provider of that service.		
	The injector is responsible for creating service instances and injecting them into classes like component		
	Injectors are inherited, which means that if a given injector can't resolve a dependency, it asks the parent injector to resolve it.		
	You can configure injectors with providers at different levels of your app, by setting a metadata value in one of three places: 1. In the @Injectable() decorator for the service itself. 2. In the @NgModule() decorator for an NgModule. 3. In the @Component() decorator for a component.		
	The @Injectable() decorator has the providedIn metadata option, where you can specify the provider of the decorated service class with the root injector, or with the injector for a specific NgModule.	@Injectable({ providedIn: 'root' })	
	The @NgModule() and @Component() decorators have the providers metadata option, where you can configure providers for NgModule-level or component-level injectors.	Questions:Possible values for provideIn ?	
	A provider can be the service class itself, so that the injector can use new to create an instance. You might also define more than one class to provide the same service in different ways, and configure different injectors with different providers.		
	Components are directives, and the providers option is inherited from @Directive(). You can also configure providers for directives and pipes at the same level as the component		
Injector hierarchy and service instances	Services are singletons within the scope of an injector. That is, there is at most one instance of a service in a given injector.		
	There is only one root injector for an app. Providing UserService at the root or AppModule level means it is registered with the root injector. There is just one UserService instance in the entire app and every class that injects UserService gets this service instance unless you configure another provider with a child injector.		
	Angular DI has a hierarchical injection system, which means that nested injectors can create their own service instances. Angular regularly creates nested injectors. Whenever Angular creates a new instance of a component that has providers specified in @Component(), it also creates a new child injector for that instance. Similarly, when a new NgModule is lazy-loaded at run time, Angular can create an injector for it with its own providers.		
	Child modules and component injectors are independent of each other, and create their own separate instances of the provided services. When Angular destroys an NgModule or component instance, it also destroys that injector and that injector's service instances.		
Dependency injection tokens	When you configure an injector with a provider, you associate that provider with a DI token. The injector maintains an internal token-provider map that it references when asked for a dependency. The token is the key to the map.		
	In simple examples, the dependency value is an instance, and the class type serves as its own lookup key. Here you get a HeroService directly from the injector by supplying the HeroService type as the token:	heroService: HeroService;	

Optional dependencies	When a component or service declares a dependency, the class constructor takes that dependency as a parameter. You can tell Angular that the dependency is optional by annotating the constructor parameter with @Optional().	import { Optional } from '@angular/core';
		constructor(@Optional() private logger: Logger) {
		  if (this.logger) {
		    this.logger.log(some_message);
		  }
		}	

	When using @Optional(), your code must be prepared for a null value. If you don't register a logger provider anywhere, the injector sets the value of logger to null.		
Hierarchical Dependency Injectors	The choices you make about where to configure providers lead to differences in the final bundle size, service scope, and service lifetime		
	When you specify providers in the @Injectable() decorator of the service itself (typically at the app root level), optimization tools such as those used by the CLI's production builds can perform tree shaking, which removes services that aren't used by your app. Tree shaking results in smaller bundle sizes.		
Platform injector	When you use providedIn:'root', you are configuring the root injector for the app, which is the injector for AppModule. The actual root of the entire injector hierarchy is a platform injector that is the parent of app-root injectors. This allows multiple apps to share a platform configuration. For example, a browser has only one URL bar, no matter how many apps you have running. The platform injector is used internally during bootstrap, to configure platform-specific dependencies. You can configure additional platform-specific providers at the platform level by supplying extraProviders using the platformBrowser() function.		
	Component-level providers configure each component instance's own injector. Angular can only inject the corresponding services in that component instance or one of its descendant component instances. Angular can't inject the same service instance anywhere else.		
	A component-provided service may have a limited lifetime. Each new instance of the component gets its own instance of the service. When the component instance is destroyed, so is that service instance.		
@Injectable-level configuration	The providedIn metadata option for a service class configures a specific injector (typically root) to use the decorated class as a provider of the service		
@NgModule-level injectors	Not working practically		
@Component-level injectors			
Element injectors	An injector does not actually belong to a component, but rather to the component instance's anchor element in the DOM. A different component instance on a different DOM element uses a different injector.	??	
	Components are special type of directive, and the providers property of @Component() is inherited from @Directive(). Directives can also have dependencies, and you can configure providers in their @Directive() metadata. When you configure a provider for a component or directive using the providers property, that provider belongs to the injector for the anchor DOM element. Components and directives on the same element share an injector.		
Injector bubbling	When a component requests a dependency, Angular tries to satisfy that dependency with a provider registered in that component's own injector. If the component's injector lacks the provider, it passes the request up to its parent component's injector. If that injector can't satisfy the request, it passes the request along to the next parent injector up the tree. The requests keep bubbling up until Angular finds an injector that can handle the request or runs out of ancestor injectors. If it runs out of ancestors, Angular throws an error.		
	If you have registered a provider for the same DI token at different levels, the first one Angular encounters is the one it uses to provide the dependency.		
When Angular creates nested injectors	Whenever Angular creates a new instance of a component that has providers specified in @Component(), it also creates a new child injector for that instance. Similarly, when a new NgModule is lazy-loaded at run time, Angular can create an injector for it with its own providers.		
	Detail explanation of Injector and Injector Tree with good practical example	https://codecraft.tv/courses/angular/dependency-injection-and-providers/ngmodule-providers-vs-component-providers-vs-component-viewproviders/	
When Injectable decorator is optional and mandatory	Refer the link	https://codecraft.tv/courses/angular/dependency-injection-and-providers/configuring/	
@Host()	You can cap the bubbling by adding the @Host() parameter decorator on the dependant-service parameter in a component's constructor. The hunt for providers stops at the injector for the host element of the component.	??	
	The class-provider syntax is a shorthand expression that expands into a provider configuration, defined by the Provider interfac	providers: [Logger] => [{ provide: Logger, useClass: Logger }]	
	The expanded provider configuration is an object literal with two properties. The provide property holds the token that serves as the key for both locating a dependency value and configuring the injector. The second property is a provider definition object, which tells the injector how to create the dependency value. The provider-definition key can be useClass, as in the example. It can also be useExisting, useValue, or useFactory.		
Alternative class providers	Different classes can provide the same service. For example, the following code tells the injector to return a BetterLogger instance when the component asks for a logger using the Logger token.	[{ provide: Logger, useClass: BetterLogger }]	
Aliased class providers	Suppose an old component depends upon the OldLogger class. OldLogger has the same interface as NewLogger, but for some reason you can't update the old component to use it. When the old component logs a message with OldLogger, you want the singleton instance of NewLogger to handle it instead. In this case, the dependency injector should inject that singleton instance when a component asks for either the new or the old logger. OldLogger should be an alias for NewLogger. If you try to alias OldLogger to NewLogger with useClass, you end up with two different NewLogger instances in your app.	[NewLogger, // Not aliased! Creates two instances of `NewLogger` { provide: OldLogger, useClass: NewLogger}]	

	To make sure there is only one instance of NewLogger, alias OldLogger with the useExisting option.	[NewLogger,
		  // Alias OldLogger w/ reference to NewLogger
		  { provide: OldLogger, useExisting: NewLogger}]	

Value providers	Sometimes it's easier to provide a ready-made object rather than ask the injector to create it from a class. To inject an object you have already created, configure the injector with the useValue option	// An object in the shape of the logger service
		export function SilentLoggerFn() {}
		
		const silentLogger = {
		  logs: ['Silent logger says "Shhhhh!". Provided via "useValue"'],
		  log: SilentLoggerFn
		};
		
		[{ provide: Logger, useValue: silentLogger }]	

Factory providers	<Example Required>		
Predefined tokens and multiple providers	<Need to Explore >		
Tree Shaking			
Tree-shakable providers	Tree shaking refers to a compiler option that removes code from the final bundle if that code not referenced in an application. When providers are tree-shakable, the Angular compiler removes the associated services from the final output when it determines that they are not used in your application. This significantly reduces the size of your bundles.		
	Ideally, if an application isn't injecting a service, it shouldn't be included in the final output. However, Angular has to be able to identify at build time whether the service will be required or not. Because it's always possible to inject a service directly using injector.get(Service), Angular can't identify all of the places in your code where this injection could happen, so it has no choice but to include the service in the injector. Thus, services provided at the NgModule or component level are not tree-shakable.		
Multiple service instances (sandboxing)	Sometimes you want multiple instances of a service at the same level of the component hierarchy. A good example is a service that holds state for its companion component instance. You need a separate instance of the service for each component. Each service has its own work-state, isolated from the service-and-state of a different component. This is called sandboxing because each service and component instance has its own sandbox to play in.	https://angular.io/guide/dependency-injection-in-action#multiple-service-instances-sandboxing	
Qualify dependency lookup with parameter decorators	The @Optional property decorator tells Angular to return null when it can't find the dependency. The @Host property decorator stops the upward search at the host component. The host component is typically the component requesting the dependency. However, when this component is projected into a parent component, that parent component becomes the host. The following example covers this second case.	https://medium.com/frontend-coach/self-or-optional-host-the-visual-guide-to-angular-di-decorators-73fbbb5c8658	
	Using the @Self decorator, the injector only looks at the component's injector for its providers. The @SkipSelf decorator allows you to skip the local injector and look up in the hierarchy to find a provider that satisfies this dependency.		
Class interface		https://angular.io/guide/dependency-injection-in-action#class-interface	

InjectionToken' objects	Use of InjectionToken ?	import { InjectionToken } from '@angular/core';
		
		export const TITLE = new InjectionToken<string>('title');
		{ provide: TITLE, useValue: 'Hero of the Month' }	

forwardRef	sometimes circular references are unavoidable. You're in a bind when class 'A' refers to class 'B' and 'B' refers to 'A'. One of them has to be defined first. The order of class declaration matters in TypeScript. You can't refer directly to a class until it's been defined.		
	The Angular forwardRef() function creates an indirect reference that Angular can resolve later. providers: [{ provide: Parent, useExisting: forwardRef(() => AlexComponent) }]	https://angular.io/guide/dependency-injection-in-action#break-circularities-with-a-forward-class-reference-forwardref	
Observables			
<Implement Onservable subscriber patterns>			
of' operator	of(...items)—Returns an Observable instance that synchronously delivers the values provided as arguments.		
from' operator	from(iterable)—Converts its argument to an Observable instance. This method is commonly used to convert an array to an observable		
<Template syntax>	HTML is the language of the Angular template. Almost all HTML syntax is valid template syntax. The <script> element is a notable exception; it is forbidden, eliminating the risk of script injection attacks. In practice, <script> is ignored and a warning appears in the browser console.		
	Some legal HTML doesn't make much sense in a template. The <html>, <body>, and <base> elements have no useful role.		
Interpolation{{...}}	You use interpolation to weave calculated strings into the text between HTML element tags and within attribute assignments.	<p>My current hero is {{currentHero.name}}</p>	
	You appear to be inserting the result between element tags and assigning it to attributes. It's convenient to think so, and you rarely suffer for this mistake. Though this is not exactly true.		
Template expressions	A template expression produces a value. Angular executes the expression and assigns it to a property of a binding target; the target might be an HTML element, a component, or a directive.		
	The interpolation braces in {{1 + 1}} surround the template expression 1 + 1.		
	JavaScript expressions that are prohibited, including: assignments (=, +=, -=, ...) new chaining expressions with ; or , increment and decrement operators (++ and --)		
Expression context	The expression context is typically the component instance.	In the following snippets, the title within double-curly braces and the isUnchanged in quotes refer to properties of the AppComponent. {{title}} <span [hidden]="isUnchanged">changed</span>	
	An expression may also refer to properties of the template's context such as a template input variable (let hero) or a template reference variable (#heroInput).	<div *ngFor="let hero of heroes">{{hero.name}}</div> <input #heroInput> {{heroInput.value}}	
	The context for terms in an expression is a blend of the template variables, the directive's context object (if it has one), and the component's members. If you reference a name that belongs to more than one of these namespaces, the template variable name takes precedence, followed by a name in the directive's context, and, lastly, the component's member names.		
Template statements	A template statement responds to an event raised by a binding target such as an element, component, or directive.		
	Template statements in the event binding section, appearing in quotes to the right of the = symbol as in (event)="statement".	<button (click)="deleteHero()">Delete hero</button>	
	Template expression supports both basic assignment (=) and chaining expressions (with ; or ,).		

	One-way
	from data source
	to view target
		{{expression}}
		[target]="expression"
		bind-target="expression"	

	One-way from view target to data source	 (target)="statement" on-target="statement"	
	Two-way	[(target)]="expression" bindon-target="expression"	
HTML attribute vs. DOM property	The distinction between an HTML attribute and a DOM property is crucial to understanding how Angular binding works. Attributes are defined by HTML. Properties are defined by the DOM (Document Object Model). A few HTML attributes have 1:1 mapping to properties. id is one example. Some HTML attributes don't have corresponding properties. colspan is one example. Some DOM properties don't have corresponding attributes. textContent is one example. Many HTML attributes appear to map to properties ... but not in the way you might think! That last category is confusing until you grasp this general rule: Attributes initialize DOM properties and then they are done. Property values can change; attribute values can't. For example, when the browser renders <input type="text" value="Bob">, it creates a corresponding DOM node with a value property initialized to "Bob". When the user enters "Sally" into the input box, the DOM element value property becomes "Sally". But the HTML value attribute remains unchanged as you discover if you ask the input element about that attribute: input.getAttribute('value') returns "Bob". The HTML attribute value specifies the initial value; the DOM value property is the current value. The disabled attribute is another peculiar example. A button's disabled property is false by default so the button is enabled. When you add the disabled attribute, its presence alone initializes the button's disabled property to true so the button is disabled. Adding and removing the disabled attribute disables and enables the button. The value of the attribute is irrelevant, which is why you cannot enable a button by writing <button disabled="false">Still Disabled</button>. Setting the button's disabled property (say, with an Angular binding) disables or enables the button. The value of the property matters. The HTML attribute and the DOM property are not the same thing, even when they have the same name.		
	In the world of Angular, the only role of attributes is to initialize element and directive state. When you write a data binding, you're dealing exclusively with properties and events of the target object. HTML attributes effectively disappear		
Attribute binding	You can set the value of an attribute directly with an attribute binding. This is the only exception to the rule that a binding sets a target property. This is the only binding that creates and sets an attribute.	<tr><td [attr.colspan]="1 + 1">One-Two</td></tr>	
	Attribute binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix attr, followed by a dot (.) and the name of the attribute. You then set the attribute value, using an expression that resolves to a string.		
Class binding	You can add and remove CSS class names from an element's class attribute with a class binding.		
	Class binding syntax resembles property binding. Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-name].	<div [class.special]="isSpecial">The class binding is special</div> Finally, you can bind to a specific class name. Angular adds the class when the template expression evaluates to truthy. It removes the class when the expression is falsy.	
Style binding		button [style.color]="isSpecial ? 'red': 'green'">Red</button>	
Built-in attribute directives	NgClass - add and remove a set of CSS classes NgStyle - add and remove a set of HTML styles NgModel - two-way data binding to an HTML form element		

NgClass		this.currentClasses = {
		    'saveable': this.canSave,
		    'modified': !this.isUnchanged,
		    'special': this.isSpecial
		  };
		<div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special</div>	

NgStyle		this.currentStyles = {
		    'font-style': this.canSave ? 'italic' : 'normal',
		    'font-weight': !this.isUnchanged ? 'bold' : 'normal',
		    'font-size': this.isSpecial ? '24px' : '12px'
		  };
		<div [ngStyle]="currentStyles"> This div is initially italic, normal weight, and extra large (24px).</div>	

ngModel	ngModelChange:	<input
		  [ngModel]="currentHero.name"
		  (ngModelChange)="setUppercaseName($event)"> is alternative of
		<input [(ngModel)]="currentHero.name">	

Template reference variables	A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a web component. Use the hash symbol (#) to declare a reference variable. You can use the ref- prefix alternative to #. The scope of a reference variable is the entire template. Do not define the same variable name more than once in the same template	<input #phone placeholder="phone number"> The #phone declares a phone variable on an <input> element. You can refer to a template reference variable anywhere in the template. The phone variable declared on this <input> is consumed in a <button> on the other side of the template <button (click)="callPhone(phone.value)">Call</button>	
	Angular sets the reference variable's value to the element on which it was declared. In the previous example, phone refers to the phone number <input> box. The phone button click handler passes the input value to the component's callPhone method. But a directive can change that behavior and set the value to something else,	https://itnext.io/working-with-angular-5-template-reference-variable-e5aa59fb9af	
ngForm	NgForm directive with the ability to track the value and validity of every control in the form.	<form (ngSubmit)="onSubmit(heroForm)" #heroForm="ngForm"> <div class="form-group"> <label for="name">Name <input class="form-control" name="name" required [(ngModel)]="hero.name"> </label> </div> <button type="submit" [disabled]="!heroForm.form.valid">Submit</button> </form> <div [hidden]="!heroForm.form.valid"> {{submitMessage}} </div>	
What behaviour/feature does ngForm add?			
Input/output		@Input() hero: Hero; @Output() deleteRequest = new EventEmitter<Hero>(); alternative is @Component({ inputs: ['hero'], outputs: ['deleteRequest'], })	
ngForm : Its usage			
ngModel : Its Feature like two way data bind, Validation	dirty,prestine, touched, untouched,valid, InValid		
The safe navigation operator (?.) and null property paths	The Angular safe navigation operator (?.) is a fluent and convenient way to guard against null and undefined values in property paths. Here it is, protecting against a view render failure if the currentHero is null.		
The non-null assertion operator (!)	??		
The $any type cast function	Sometimes a binding expression will be reported as a type error and it is not possible or difficult to fully specify the type. To silence the error, you can use the $any cast function to cast the expression to the any type.	<div> The hero's marker is {{$any(hero).marker}} </div>	
Angular Elements	<Can expolre at leter stage>		
Dynamic Component	<Can expolre at leter stage>		
Directive	There are three kinds of directives in Angular: 1. Components—directives with a template. 2. Structural directives—change the DOM layout by adding and removing DOM elements. 3. Attribute directives—change the appearance or behavior of an element, component, or another directive. Structural Directives change the structure of the view. Two examples are NgFor and NgIf. Attribute directives are used as attributes of elements.		
Attribute directive	It's the brackets ([]) that make it an attribute selector. Angular locates each element in the template that has an attribute named appHighlight and applies the logic of this directive to that element.	@Directive({ selector: '[appHighlight]' })	

	The import statement specifies an additional ElementRef symbol from the Angular core library:
	
	You use the ElementRef in the directive's constructor to inject a reference to the host DOM element, the element to which you applied appHighlight.
	
	ElementRef grants direct access to the host DOM element through its nativeElement property.
		import { Directive, ElementRef } from '@angular/core';
		constructor(el: ElementRef) {
		    el.nativeElement.style.backgroundColor = 'yellow';
		  }
		Do we need to add behaviour to the construction only ?
		When constructor will be executed?	

Structural Directives	Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements. As with other directives, you apply a structural directive to a host element. The directive then does whatever it's supposed to do with that host element and its descendants. Structural directives are easy to recognize. An asterisk (*) precedes the directive attribute name as in this example.	<div *ngIf="hero" class="name">{{hero.name}}</div>	
	Throughout this guide, you'll see a directive spelled in both UpperCamelCase and lowerCamelCase. Already you've seen NgIf and ngIf. There's a reason. NgIf refers to the directive class; ngIf refers to the directive's attribute name.		
	The *ngIf directive moved to the <ng-template> element where it became a property binding,[ngIf]. The rest of the <div>, including its class attribute, moved inside the <ng-template> element.	<div *ngIf="hero" class="name">{{hero.name}}</div> to <ng-template [ngIf]="hero"> <div class="name">{{hero.name}}</div> </ng-template>	
	One structural directive per host element		
	<Example to create a custom directive>		

Pipe	This pipe definition reveals the following key points:
	•A pipe is a class decorated with pipe metadata.
	•The pipe class implements the PipeTransform interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.
	•There will be one additional argument to the transform method for each parameter passed to the pipe. Your pipe has one such parameter: the exponent.
	•To tell Angular that this is a pipe, you apply the @Pipe decorator, which you import from the core Angular library.
	•The @Pipe decorator allows you to define the pipe name that you'll use within template expressions. It must be a valid JavaScript identifier.
		import { Pipe, PipeTransform } from '@angular/core';
		'@Pipe({name: 'exponentialStrength'})
		export class ExponentialStrengthPipe implements PipeTransform {
		 transform(value: number, exponent: string): number {
		  let exp = parseFloat(exponent);
		  return Math.pow(value, isNaN(exp) ? 1 : exp);
		 }
		}	

	There are two categories of pipes: pure and impure. Pipes are pure by default.		
Pure pipes	Angular executes a pure pipe only when it detects a pure change to the input value. A pure change is either a change to a primitive input value (String, Number, Boolean, Symbol) or a changed object reference (Date, Array, Function, Object). Angular ignores changes within (composite) objects. It won't call a pure pipe if you change an input month, add to an input array, or update an input object property. This may seem restrictive but it's also fast. An object reference check is fast—much faster than a deep check for differences—so Angular can quickly determine if it can skip both the pipe execution and a view update.		
Impure pipes	Angular executes an impure pipe during every component change detection cycle. An impure pipe is called often, as often as every keystroke or mouse-move. With that concern in mind, implement an impure pipe with great care. An expensive, long-running pipe could destroy the user experience.		
Lifecycle sequence	ngOnChanges() :Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. Called before ngOnInit() and whenever one or more data-bound input properties change. Angular only calls the hook when the value of the input property changes. Angular doesn't care that the object's own property property changed.		
	ngOnInit() Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called once, after the first ngOnChanges().		
	ngDoCheck() Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().		
	ngAfterContentInit() Respond after Angular projects external content into the component's view / the view that a directive is in. Called once after the first ngDoCheck().		
	ngAfterContentChecked() Respond after Angular checks the content projected into the directive/component. Called after the ngAfterContentInit() and every subsequent ngDoCheck().		
	ngAfterViewInit() Respond after Angular initializes the component's views and child views / the view that a directive is in. Called once after the first ngAfterContentChecked().		
	ngAfterViewChecked() Respond after Angular checks the component's views and child views / the view that a directive is in. Called after the ngAfterViewInit and every subsequent ngAfterContentChecked().		
	ngOnDestroy() Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called just before Angular destroys the directive/component.		
OnInit()	Use ngOnInit() for two main reasons: 1.To perform complex initializations shortly after construction. 2.To set up the component after Angular sets the input properties		
	Don't fetch data in a component constructor. You shouldn't worry that a new component will try to contact a remote server when created under test or before you decide to display it. Constructors should do no more than set the initial local variables to simple values. An ngOnInit() is a good place for a component to fetch its initial data. Remember also that a directive's data-bound input properties are not set until after construction. That's a problem if you need to initialize the directive based on those properties. They'll have been set when ngOnInit() runs.		

OnDestroy()	Put cleanup logic in ngOnDestroy(), the logic that must run before Angular destroys the directive. This is the time to notify another part of the application that the component is going away. This is the place to free resources that won't be garbage collected automatically. Unsubscribe from Observables and DOM events. Stop interval timers. Unregister all callbacks that this directive registered with global or application services. You risk memory leaks if you neglect to do so.		
Content projection	Content projection is a way to import HTML content from outside the component and insert that content into the component's template in a designated spot. AngularJS developers know this technique as transclusion.	https://dzone.com/articles/simplifying-content-projection-in-angular	
@ViewChild	<Explore more to understand life cycle hook>		
@ContentChild	<Explore more to understand life cycle hook>		
AbstractControl in input parameter	<Explore>		
SimpleChanges Type	<Explore>		
	View queries are set before the ngAfterViewInit callback is called.		
Supported selectors include:	any class with the @Component or @Directive decorator a template reference variable as a string (e.g. query <my-component #cmp></my-component> with@ViewChild('cmp')`) any provider defined in the child component tree of the current component (e.g. @ViewChild(SomeService) someService: SomeService) any provider defined through a string token (e.g. @ViewChild('someToken') someTokenVal: any) a TemplateRef (e.g. query <ng-template></ng-template> with @ViewChild(TemplateRef) template;)	??	
<base href> :	1. as the first child in the <head> tag 2. to tell the router how to compose navigation URLs. 3. If the app folder is the application root, as it is for the sample application, set the href value as "/"		
Why Angular 2	Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript.		
decorator	In TypeScript, you attach metadata by using a decorator.	Component, directive, Injectable, Input, are the example of predefined decorator.	
Metadata	Metadata tells Angular how to process a class.		

Why dependency injection?	Car class creates everything it needs inside its constructor. What's the problem? The problem is that the Car class is brittle, inflexible, and hard to test.
	This Car needs an engine and tires. Instead of asking for them, the Car constructor instantiates its own copies from the very specific classes Engine and Tires.
	What if the Engine class evolves and its constructor requires a parameter? That would break the Car class and it would stay broken until you rewrote it along the lines of this.engine = new Engine(theNewParameter). The Engineconstructor parameters weren't even a consideration when you first wrote Car. You may not anticipate them even now. But you'll have to start caring because when the definition of Engine changes, the Car class must change. That makes Car brittle.
	What if you want to put a different brand of tires on your Car? Too bad. You're locked into whatever brand the Tires class creates. That makes the Car class inflexible.
	Right now each new car gets its own engine. It can't share an engine with other cars. While that makes sense for an automobile engine, surely you can think of other dependencies that should be shared, such as the onboard wireless connection to the manufacturer's service center. This Carlacks the flexibility to share services that have been created previously for other consumers.
		export class Car {
		public engine: Engine;
		public tires: Tires;
		public description = 'No DI';
		constructor() {
		this.engine = new Engine();
		this.tires = new Tires();
		}
		// Method using the engine and tires
		drive() {
		return `${this.description} car with ` +
		`${this.engine.cylinders} cylinders and ${this.tires.make} tires.`;
		}
		}	

	It's a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.		
When to use NgModule versus an application component to register provider	On the one hand, a provider in an NgModule is registered in the root injector. That means that every provider registered within an NgModule will be accessible in the entire application. On the other hand, a provider registered in an application component is available only on that component and all its children.		
Why @Injectable()?	@Injectable() marks a class as available to an injector for instantiation. Generally speaking, an injector reports an error when trying to instantiate a class that is not marked as @Injectable()		
injector in detail	https://angular.io/docs/ts/latest/guide/dependency-injection.html		
super-short primer on Angular's template syntax	[property]="expression": set property of an element to the value of expression (event)="statement": execute statement when event occurred [(property)]="expression": create two-way binding with expression [class.special]="expression": add special CSS class to element when the value of expression is truthy [style.color]="expression": set color CSS property to the value of expression		
@Injectable()	The @Injectable() decorator tells TypeScript to emit metadata about the service. The metadata specifies that Angular may need to inject other dependencies into this service. Although the HeroService doesn't have any dependencies at the moment, applying the @Injectable() decorator from the start ensures consistency and future-proofing.		
Don't use new with the HeroService	You could create a new instance of the HeroService with new like this:	heroService = new HeroService(); // don't do this	
Promise	Promises are a pattern that helps with one particular kind of asynchronous programming: a function (or method) that returns a single result asynchronously. One popular way of receiving such a result is via a callback ("callbacks as continuations")	asyncFunction(arg1, arg2, result => { console.log(result); });	
	Promises provide a better way of working with callbacks: Now an asynchronous function returns a Promise, an object that serves as a placeholder and container for the final result. Callbacks registered via the Promise method then() are notified of the result:	asyncFunction(arg1, arg2) .then(result => { console.log(result); });	
Promises have the following advantages	No inversion of control: similarly to synchronous code, Promise-based functions return results, they don't (directly) continue – and control – execution via callbacks. That is, the caller stays in control. Chaining is simpler: If the callback of then() returns a Promise (e.g. the result of calling another Promise-based function) then then() returns that Promise (how this really works is more complicated and explained later). As a consequence, you can chain then() method calls:		
	Composing asynchronous calls (loops, mapping, etc.): is a little easier, because you have data (Promise objects) you can work with. Error handling: As we shall see later, error handling is simpler with Promises, because, once again, there isn't an inversion of control. Furthermore, both exceptions and asynchronous errors are managed the same way. Cleaner signatures: With callbacks, the parameters of a function are mixed; some are input for the function, others are responsible for delivering its output. With Promises, function signatures become cleaner; all parameters are input. Standardized: Prior to Promises, there were several incompatible ways of handling asynchronous results (Node.js callbacks, XMLHttpRequest, IndexedDB, etc.). With Promises, there is a clearly defined standard: ECMAScript 6. ES6 follows the standard Promises/A+ [1]. Since ES6, an increasing number of APIs is based on Promises.		
Lazy-loading modules with the router	A lazy-loaded module location is a string, not a type. In this app, the string identifies both the module file and the module class, the latter separated from the former by a #.	{ path: 'crisis', loadChildren: 'app/crisis/crisis.module#CrisisModule' }	
	forRoot and forChild are conventional names for methods that deliver different import values to root and feature modules.	Never call RouterModule.forRoot in a feature-routing module. Always call RouterModule.forChild in a feature-routing module.	
	For this reason Angular provides a way to separate providers out of the module so that same module can be imported into the root module with providers and child modules without providers.		
RxJs			

Concepts	Discription	Example
Introduction	• DATABASE - A set of inter-related data • DBMS - A software that manages the data • SCHEMA - A set of structures and relationships, which meet a specific need	
DataBase Model	• Flat Model – Data is stored in an array of two dimensions • Hierarchical model – Data and the relationships among them are represented in the form of a tree structure ,. • Network model – Data and the relationships among them are represented in the form of records and links. • Relational model – Data is stored in tables and the relationship among them is represented in common column called foreign key	
Normalization	Process of efficiently organizing data in a database eliminate redundant data ensure data dependencies make sense	
Fisrt normalized Form	A relation is in first normal form if the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain	http://en.wikipedia.org/wiki/First_normal_form
Second normalized Form	a table is in 2NF if and only if it is in 1NF and no non-prime attribute is dependent on any proper subset of any candidate key of the table. A non-prime attribute of a table is an attribute that is not a part of any candidate key of the table. If in table single column is not suffucient to identitfy the unique row and more than one column is required and other colums deend the part of comosites key then 2NF is required,	http://en.wikipedia.org/wiki/Second_normal_form
Third normalized Form	a table is in 3NF if and only if 1.the entity is in second normal form and 2.all the attributes in a table are dependent on the primary key and only the primary key	http://en.wikipedia.org/wiki/Third_normal_form
Data Integrity	Entity Integrity: Entity integrity ensures that no records are duplicated --PrimaryKey Domain Integrity: Domain integrity is the validity of entries for a given column --can be enforced by type(through data types),format (through CHECK constraints and rules), or the range of possible values(through FOREIGN KEY constraints, CHECK constraints, DEFAULT definitions,NOT NULL definitions, and rules). :Referential integrity preserves the defined relationships between tables Data Integrity or Referential Integrity: when records are entered or deleted User-defined integrity:Refers to a set of rules specified by a user, which do not belong to the entity, domain, and referential integrity categories	
	• The Data Definition Language (DDL) • Data Manipulation Language (DML) • Data Control Language (DCL) • Transactional Control Language (TCL)	
DDL :	• It is the subset of SQL which contains the commands used to create alter and destroy databases and database objects • DDL includes the commands for handling tasks such as creating tables, indexes, views, and constraints • The commands are – CREATE – ALTER – DROP – [USE]	
DML:	It is the subset of SQL used to access and manipulate data contained within the data structures previously defined via DDL • The Commands are: – INSERT - Adds data to a database. – UPDATE - Modifies data in a database. – DELETE - Removes data from a database. – SELECT- Retrieves data from a database. – TRUNCATE – Delete All	
DCL:	• It is the subset of SQL Commands that control a database, including administering privileges and committing data • It is used to create roles, permissions, and to control access to database. • The Commands are – Grant – Revoke – [Deny]	*<Example>*
	Grant is used to provide permissions like Select, All, Execute to user on the database objects like Tables, Views, Databases etc.	Syntax: Grant privilageName on objectName To{userName/Public/roleName} [with Grant Option] E.g: grant select on deep to user24
	With Grant Option: if you use the WITH GRANT option, then user24 can GRANT SELECT privilege on the Deep table to another user. Later, if you REVOKE the SELECT privilege on employee from user24, still user25 will have SELECT privilege on the table	
	Revoke: Revoke is used to remove the permissions or privileges provided to a user by the Grant command	Revoke privilageName on objectName from{userName/public/roleName} E.g.: revoke select on eep from public
TCL:	• It is used to manage different transactions occurring within a database. • The commands are – COMMIT – ROLLBACK – [Save Tran]	*<Example>*
SystemDB:	Master model tempdb msdb distribution	
master	The master database hold all of the information related to logins, endpoints, linked servers, and user databases, it’s important that you take a backup of the master database after configuring any of these server level changes. Otherwise, if your SQL Server suffers a catastrophic failure, those changes will be lost to the sands of time	<Need more and concrete information of all system databases>
model	The model database is used as the template for all databases created on an instance of SQL Server. Because tempdb is created every time SQL Server is started, the model database must always exist on a SQL Server system. The entire contents of the model database, including database options, are copied to the new database. Some of the settings of model are also used for creating a new tempdb during start up, so the model database must always exist on a SQL Server system.	
msdb	msdb is used by the SQL Server Agent, database mail, Service Broker, and other services. If you aren’t actively working with things like jobs, alerts, log shipping, etc you can pretty safely ignore msdb… sort of. One important item is that msdb holds backup history. Using the msdb tables (you can start by taking a look at msdb.dbo.backupset), it’s possible to determine when each database and filegroup was last backed up.	
Resource database	The resource database is a hidden system database. This is where system objects are stored. It isn’t possible to see the resource database by normal means. However you can see the data file by navigating to C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Binn. The exact size and modification data of this file will be different from version to version, but the modified date should be the same date that you see when you run SELECT @@version. It is best to think of the resource database as if it were another system DLL. The resource database is designed to make it easy for quick database upgrades. If new system objects are being put in place, it is only necessary to swap out the resource database MDF file. Typically, the only way to view the contents of the resource database is using the OBJECT_DEFINITION system function.	

Concepts	Discription	Example
tempdb	We come, at last, to tempdb. Tempdb is the workhorse of the system databases. It is the workspace that SQL Server uses to store the intermediate results of query processing and sorting. You know how you see those spools in your execution plans? When you see one of those, SQL Server is probably spooling the data to a temporary table in the tempdb. Outside of storing temporary results, tempdb is also used during snapshot isolation and for user created temporary tables (this includes table variables). One thing that is interesting to note about tempdb is that it is re-created every time the SQL Server service is started. Any objects that you have created in tempdb will be gone once the server restarts. If you want specific tables or stored procedures to always be available in tempdb, you will need to add them to the model database or else use a stored procedure to create them in tempdb when the SQL Server is started.	
System catalog	??	
database catalog	??	
SQL_Variant	sql_variant - A data type that stores values of various SQL Server-supported data types, except varchar, nvarchar, text, ntext, timestamp, and sql_variant.	https://docs.microsoft.com/en-us/sql/t-sql/data-types/sql-variant-transact-sql?view=sql-server-2017
XML datatype and Typed XML datatype		
Difference between char,varchar and nvarchar	Char datatype which is used to store fixed length of characters. Suppose if we declared char(50) it will allocates memory for 50 characters. Once we declare char(50) and insert only 10 characters of word then only 10 characters of memory will be used and other 40 characters of memory will be wasted. Varchar means variable characters and it is used to store non-unicode characters. It will allocate the memory based on number characters inserted. Suppose if we declared varchar(50) it will allocates memory of 0 characters at the time of declaration. Once we declare varchar(50) and insert only 10 characters of word it will allocate memory for only 10 characters.It takes 1 byte per character nvarchar datatype same as varchar datatype but only difference nvarchar is used to store Unicode characters and it allows you to store multiple languages in database.It takes 2 bytes per Unicode/Non-Unicode	http://sqlhints.com/2011/12/23/difference-between-varchar-and-nvarchar/
User-defined Data Types	If a user-defined data type is created in the model database, it exists in all new user-defined databases. However, if the data type is created in a user-defined database, the data type exists only in that user-defined database.	EXEC sp_addtype city, 'nvarchar(15)', NULL EXEC sp_droptype city
All basic Queries	Creating table with primary and foreigh key Add primary key and foreign query later droping table add/delete/Modify column after creating table..	ALTER TABLE Employee Add city nvarchar(max)
IDENTITY	Only one identifier column and one globally unique identifier column can be created for each table. @@identity to determine most recent value	
key word @@IDENTITY	Is a system function that returns the last-inserted identity value.	
Database SCHEMA	A database schema is a visual and logical architecture of a database created on a database management system. It provides a graphical view of the entire database architecture and structure. It provides a means for logically grouping and displaying database objects such as tables, fields, functions and relations.	
	A database schema typically shows the different tables, their fields and the relationship between them and other tables. The database schema is defined within the native database language; therefore the logical structure and visualization of schema might vary in each database language. It helps database administrators in understanding the architectural layout of the database. In addition to tables and fields, a database schema defines a database's: Indexes Views Triggers Database links Events Procedures Functions	
		CREATE TABLE my_publishers (pub_id char(4) , pub_name varchar(40), constraint my_chk_constraint check (pub_id in ("1234", “4321", “1212") or pub_name not like "Bad Books"))
Check Constrain		ALTER TABLE dbo.Employees ADD CONSTRAINT CK_birthdate CHECK (BirthDate > '01-01-1900' AND BirthDate < getdate())
Default Constrain		ALTER TABLE dbo.Customers ADD CONSTRAINT DF_contactname DEFAULT 'UNKNOWN' FOR ContactName
PrimaryKey		ALTER TABLE dbo.Customers ADD CONSTRAINT PK_Customers PRIMARY KEY NONCLUSTERED (CustomerID)
UNIQUE		ALTER TABLE dbo.Suppliers ADD CONSTRAINT U_CompanyName UNIQUE NONCLUSTERED (CompanyName)
FOREIGN KEY		ALTER TABLE dbo.Orders ADD CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID) REFERENCES dbo.Customers(CustomerID)
WITH NOCHECK	Use if existing data will not change	
		ALTER TABLE dbo.Employees WITH NOCHECK ADD CONSTRAINT FK_Employees_Employees FOREIGN KEY (ReportsTo) REFERENCES dbo.Employees(EmployeeID)
Default	sp_binddefault It is database specific	CREATE DEFAULT phone_no_default AS '(000)000-0000' GO EXEC sp_bindefault phone_no_default, 'Customers.Phone'
RULE	sp_bindrule It is database specific	CREATE RULE regioncode_rule AS @regioncode IN ('IA', 'IL', 'KS', 'MO') GO EXEC sp_bindrule regioncode_rule, 'Customers.Region'
	order is fixed WGHO	SELECT [DISTINCT][TOP n] <columns > [FROM] <table names> [WHERE] <criteria that must be true for a row to be chosen> [GROUP BY] <columns for grouping aggregate functions> [HAVING] <criteria that must be met for aggregate functions> [ORDER BY] <optional specification of how the results should be sorted>
		Select CONVERT(char,100) --converts 100 to ‘100’ Select CAST(100 as char)
Cast and Convert Difference	Parameter CAST CONVERT ANSI standard Yes No Data-type coverage Limited Full (Date & Date Time values supported) Performance No difference No difference Microsoft SQL CONVERT CONVERT Server implementation	http://beyondrelational.com/modules/2/blogs/77/posts/11334/cast-vs-convert-is-there-a-difference-as-far-as-sql-server-is-concerned-which-is-better.aspx . Select CONVERT(char,100) --converts 100 to ‘100’ Select CAST(100 as char)
DISTINCT	DISTINCT is used to eliminate duplicate rows	SELECT DISTINCT Region - lists out all regions in which employees live FROM Northwind.dbo.Employees
Diff between Count(columnname) and count(*)	COUNT(*) returns a count of all records COUNT(table.ColumnName) returns a count of all non-null values.	
Groupby Difference	Columns in a select list must be in the group by expression or they must be arguments of aggregate functions	

Concepts	Discription	Example		
Having Clause	Columns in a select list must be in the group by expression or they must be arguments of aggregate functions.	Order : The where clause excludes rows that do not meet its search conditions. The group by clause collects the remaining rows into one group for each unique value in the group by expression. Aggregate functions specified in the select list calculate summary values for each group. The having clause excludes rows from the final results that do not meet its search conditions.		
JOIN	1.INNER JOIN : (Equi JOIN) 2.Outer JOIN : LEFT JOIN RIGHT JOIN FULL JOIN 3.SELF JOIN 4.CROSS JOIN			
SET operators	UNION:Combine two or more result sets into a single set, without duplicates. UNION ALL:Combine two or more result sets into a single set, including all duplicates. INTERSECT:Takes the data from both result sets which are in common. EXCEPT:Takes the data from first result set, but not the second (i.e. no matching to each other)			
SET operators	In order to sort the result, an ORDER BY clause should be part of the last statement.			
Restriction of SubQuery	Inner query should not contain order by.	*Explore*		
COMPUTE BY,COMPUTE				
DELETE,TRUNCATE,DROP	DROP: drop the table data with table defination. DELETE:can delete the data as per condition. But it does not free the space containing the table. Truncate: delete all the table data.free the space containing the table. TRUNCATE TABLE is similar to the DELETE statement with no WHERE clause	DROP _tableName DELETE TABLE FROM _tableName where <condition> TRUNCATE TABLE _tableName		
Difference Between TRUNCATE, DELETE, And DROP In SQL	TRUNCATE is a DDL command TRUNCATE is executed using a table lock and whole table is locked for remove all records. We cannot use WHERE clause with TRUNCATE. TRUNCATE removes all rows from a table. Minimal logging in transaction log, so it is faster performance wise. TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log. Identify column is reset to its seed value if table contains any identity column. To use Truncate on a table you need at least ALTER permission on the table. Truncate uses less transaction space than the Delete statement. Truncate cannot be used with indexed views. TRUNCATE is faster than DELETE.			
	DELETE is a DML command. DELETE is executed using a row lock, each row in the table is locked for deletion. We can use where clause with DELETE to filter & delete specific records. The DELETE command is used to remove rows from a table based on WHERE condition. It maintain the log, so it slower than TRUNCATE. The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row. Identity of column keep DELETE retains the identity. To use Delete you need DELETE permission on the table. Delete uses the more transaction space than Truncate statement. Delete can be used with indexed views. To execute a DELETE queue, delete permissions are required on the target table. If you need to use a WHERE clause in a DELETE, select permissions are required as well.			
	The DROP command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back. DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command. DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back		https://www.c-sharpcorner.com/blogs/difference-between-truncate-delete-and-drop-in-sql-server1	
Cluster Index	Unique Clustered index is automatically created when column has a PRIMARY KEY constraint	CREATE CLUSTERED INDEX CL_lastname ON employees(lastname) DROP INDEX employees.CL_lastname		
	Unique non clustered index is automatically created when a column has UNIQUE constraint	CREATE UNIQUE NONCLUSTERED INDEX U_CustID ON customers(CustomerID)		
sp_helpindex	Will shows index details of table	EXEC sp_helpindex Customers		

Views	– To hide the complexity of the underlying database schema, or customize the data and schema for a set of users.
	– To control access to rows and columns of data.
	
	Objective of creating views is Abstraction , not performance
	
	Restrictions on View Definitions
	– Cannot include ORDER BY clause
	– Cannot include INTO keyword
		CREATE VIEW dbo.OrderSubtotalsView (OrderID, Subtotal) -- O/P Column name
		AS
		SELECT OD.OrderID,
		SUM(CONVERT(money,(OD.UnitPrice*Quantity*(1-Discount)/100))*100)
		FROM [Order Details] OD
		GROUP BY OD.OrderID
		GO		

Limitation of View	You cannot pass parameters to SQL Server views Cannot use an Order By,COMPUTE, or COMPUTE BY or Into clause with views without specifying FOR XML or TOP Views cannot be created on Temporary Tables You cannot associate rules and defaults with views You can build views on other views and on procedures that reference views. Only INSTEAD OF triggers can be associated with views. When a table or view is dropped, any views in the same database are also dropped. It is not possible to create an index on a view To alter a view, it must be dropped and re-created. Could not renamed or change the output column name			
Types of Views	• Standard Views • Indexed Views • Partitioned Views			
Rules for insert or update in views			https://www.codeproject.com/Articles/236425/How-to-Insert-Data-Using-SQL-Views-Created-Using-M	

WITH ENCRYPTION	Keyword WITH ENCRYPTION is used to encrypt the text of the Stored Procedure. One SP are encrypted it is not possible to get original text of the SP from SP itself. User who created SP will need to save the text to be used to create SP somewhere safe to reuse it again	CREATE VIEW dbo.[Order Subtotals] WITH ENCRYPTION
		AS
		SELECT OrderID,
		Sum(CONVERT(money,(UnitPrice*Quantity*(1-Discount)/100))*100)
		AS Subtotal
		FROM [Order Details		

WITH RECOMPILE on stored procedure	With Recompile at Stored proc level,will cause recompilation every time the proc is executed and query is not saved to Cache			
<Advanced Topic> *OPTION (RECOMPILE) and* *before or after OPTION (RECOMPILE)* *Constant Folding* *avoiding Parameter Sniffing* *OPTIMIZE FOR* *WITH RECOMPILE (In Detail)*	Go through the link for better understanding	https://blogs.msdn.microsoft.com/robinlester/2016/08/10/improving-query-performance-with-option-recompile-constant-folding-and-avoiding-parameter-sniffing-issues/		
Global Variable	• @@ERROR • @@FETCH_STATUS • @@IDENTITY • @@ROWCOUNT • @@SERVERNAME • @@SPID • @@TRANCOUNT • @@VERSION	https://www.codeproject.com/Articles/39131/Global-Variables-in-SQL-Server		

Concepts	Discription	Example		
Cursor	– Is a data structure which helps in defining a result set and perform a complex business logic on each row of the result set – A cursor can be viewed as a pointer to one row in a set of rows – The main advantage is that we can process the data row-by -row. – They are NOT database objects NOTE : cursors are the SLOWEST way to access data inside SQL Server . Therefore they used be used only when there is an absolute need			
How cursors work	A cursor is a symbolic name associated with a select statement. It consists of the following parts: Cursor result set - the set (table) of rows resulting from the execution of a query that is associated with the cursor. Cursor position - a pointer to one row within the cursor result set. The cursor position indicates the current row of the cursor. You can explicitly modify or delete that row using update or delete statements with a clause naming the cursor			
	1. Declaring the cursor :If you declare the cursor within a stored procedure, Server creates the cursor structure and compiles the query defined for that cursor. It stores the compiled query plan but does not execute it. 2. Opening the cursor :When the cursor is opened, Server needs to perform preliminary operations for executing a scan and returning a result set. 3. Fetching from the cursor: The fetch command executes the compiled cursor to return one or more rows meeting the conditions defined in the cursor 4. Processing the row by examining, updating, or deleting it through the cursor : 5. Closing the cursor:Server closes the cursor result set, removes any remaining temporary tables, and releases the server resources held for the cursor structure. However, it keeps the query plan for the cursor so that it can be opened again 6. Deallocating the cursor: Server dumps the query plan from memory and eliminates all trace of the cursor structure	DECLARE @USER_ID INT DECLARE @USER_NAME VARCHAR(10) DECLARE @TABLE_COUNT INT DECLARE dbuser_cursor CURSOR READ_ONLY FOR SELECT uid,name from sysusers OPEN dbuser_cursor FETCH NEXT FROM dbuser_cursor INTO @USER_ID,@USER_NAME WHILE @@FETCH_STATUS = 0 BEGIN print CAST(@USER_ID as char(2))+' '+@USER_NAME SELECT TYPE,COUNT(ID) FROM SYSOBJECTS where uid=@USER_ID GROUP BY TYPE FETCH NEXT FROM dbuser_cursor INTO @USER_ID,@USER_NAME END CLOSE dbuser_cursor DEALLOCATE dbuser_cursor		
Types of cursors	1. static :whenever cursor opens all records will be stored in tempdb. changes on the data can't be seen. 2. Key setdriven:Whenever cursor opens only key values will be stored in the cursor If we use fetch next it will take key value and search record from the database. All committed writes are visible. 3.Dynamic cursors : Cursor can move anywhere and all the changes on the data can be viewed. 4. forward-only :Cursor moves one step forward. Can't move backwards.			
StoreProcedure types	1.System 2.Temporary 3.Extended			
Store Procedure	Two types of parameters 1. Input type : IN 2. Output type : OUT			
Exception is SP	Error_message(), Error_number(), Error_severity(), Error_state(), Error_line(), Error_procedure()			
Exception is SP	RAISERROR ()			
Recompilation of SP	SP needs recompilation when – Data in underlying tables are changed ----- ?? – Indexes are added /removed in tables	– CREATE PROCEDURE [WITH RECOMPILE] – EXECUTE [procedure]WITH RECOMPILE] – sp_recompile [procedure]		
return value in SP	– OUTPUT parameter • More than 1 parameter can be of type OUTPUT – Return statement • Used to provide the execution status of the procedure to the calling program • Only one value can be returned • 0 to -99 are reserved for internal usage , one can return customized values also • Return value can be processed by the calling program as	exec @return_value = <storedprocname>		
Difference between procedure and function	Procedures •Return single integer value represents return status • Use execute (EXEC) statement to execute stored procedure •Use output parameter to pass values to caller	Function • Return single value of any scalar data type supported by SQL server 2005 or Table type • Can be called through select statement if it returns scalar value otherwise can be called through from statement if it returns table. • Use return statement to pass values to caller		
	Basic Difference 01.Function must return a value but in Stored Procedure it is optional(Procedure can return zero or n values). 02.Functions can have only input parameters for it whereas Procedures can have input/output parameters . 03.Functions can be called from Procedure whereas Procedures cannot be called from Function.			
	Advance Difference 01.Procedure allows SELECT as well as DML (INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it. 02.Procedures can not be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement. 03.Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be. 04.The most important feature of stored procedures over function is to retention and reuse the execution plan while in case of function it will be compiled every time. 05.Functions that return tables can be treated as another rowset. This can be used in JOINs with other tables. 06.Inline Function can be though of as views that take parameters and can be used in JOINs and other Rowset operations. 07.Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function. 08.We can go for Transaction Management in Procedure whereas we can't go in Function.			
Save Point in Transaction		http://www.blackwasp.co.uk/SQLSavepoints.aspx		
can we have two commit statement in single transaction	??			
Implicit transction	No need to write "Begin Transaction/Tran" SET IMPLICIT_TRANSACTIONS ON the instance of the Database Engine automatically starts a transaction when it first executes T-sql query	http://technet.microsoft.com/en-us/library/ms188317(v=sql.105).aspx		
Explicit transaction	Need to mention the start of the transaction.			

Concepts	Discription	Example		
Triggers	A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updation of data. It is a database object which is bound to a table and is executed automatically. You can't explicitly invoke triggers. The only way to do this is by performing the required action no the table that they are assigned to.	http://www.codeproject.com/Articles/25600/Triggers-SQL-Server		
Types of triggers	1.After trigger 2.Instead-of Triggers			
1.After trigger	These triggers run after an insert, update or delete on a table. They are not supported for views. AFTER TRIGGERS can be classified further into three types as: a. AFTER INSERT Trigger. b. AFTER UPDATE Trigger. c. AFTER DELETE Trigger.			

AFTER INSERT Trigger.	The FOR INSERT specifies that this is an AFTER INSERT trigger. In place of FOR INSERT, AFTER INSERT can be used. Both of them mean the same.
	
	In the trigger body, table named inserted has been used. This table is a logical table and contains the row that has been inserted
		CREATE TRIGGER trgAfterInsert ON [dbo].[Employee_Test]
		FOR INSERT
		AS
		  SELECT * from inserted		

AFTER UPDATE Trigger	This trigger is fired after an update on the table.
	There is no logical table updated like the logical table inserted. We can obtain the updated value of a field from the update (column_name) function. In our trigger,
		CREATE TRIGGER trgAfterUpdate ON [dbo].[Employee_Test]
		FOR UPDATE
		AS
		  SELECT * from inserted
		if update(Emp_Name)		

AFTER DELETE Trigger	This trigger is fired after a delete on the table.
	In this trigger, the deleted record's data is picked from the logical deleted table
		CREATE TRIGGER trgAfterDelete ON [dbo].[Employee_Test]
		AFTER DELETE
		AS
		SELECt * FROM deleted;		

Instead Of Triggers	These can be used as an interceptor for anything that anyone tried to do on our table or view. If you define an Instead Of trigger on a table for the Delete operation, they try to delete rows, and they will not actually get deleted (unless you issue another delete instruction from within the trigger) INSTEAD OF TRIGGERS can be classified further into three types as: a.INSTEAD OF INSERT Trigger. b. INSTEAD OF UPDATE Trigger. c. INSTEAD OF DELETE Trigger.	CREATE TRIGGER trgInsteadOfDelete ON [dbo].[Employee_Test] INSTEAD OF DELETE AS		
DDL triggers.				
Logon triggers				
Set NOCOUNT	Stops the message that shows the count of the number of rows affected by a Transact-SQL statement or stored procedure from being returned as part of the result set			
SET ANSI NULLS ON	When SET ANSI_NULLS is ON, a SELECT statement that uses WHERE column_name = NULL returns zero rows even if there are null values in column_name. A SELECT statement that uses WHERE column_name <> NULL returns zero rows even if there are nonnull values in column_name. When SET ANSI_NULLS is OFF, the Equals (=) and Not Equal To (<>) comparison operators do not follow the ISO standard. A SELECT statement that uses WHERE column_name = NULL returns the rows that have null values in column_name. A SELECT statement that uses WHERE column_name <> NULL returns the rows that have nonnull values in the column.	By default it will be ON. And use can use IS NULL or IS NOT NULL		
If we add two column in index ..use ?				
incoude in index				
differnce between temp table and table datatype	⇒ Table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if there is a memory pressure the pages belonging to a table variable may be pushed to tempdb. ⇒ Table variables cannot be involved in transactions, logging or locking. This makes @table faster then #temp. So table variable is faster then temporary table. ⇒ Temporary tables are allowed CREATE INDEXes whereas, Table variables aren't allowed CREATE INDEX instead they can have index by using Primary Key or Unique Constraint. ⇒ Table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables. ⇒ Temporary tables are visible in the created routine and also in the child routines. Whereas, Table variables are only visible in the created routine. ⇒ Temporary table allows Schema modifications unlike Table variables.		https://www.c-sharpcorner.com/article/temporary-tables-and-table-variables-in-sql/	
types of function				
with(nolock)				

Merge	In SQL Server 2008, you can perform insert, update, or delete operations in a single statement using the MERGE statement. The MERGE statement allows you to join a data source with a target table or view, and then perform multiple actions against the target based on the results of that join.	DECLARE @TargetTable TABLE(EmployeeID int, EmployeeName varchar(10))
		DECLARE @SourceTable TABLE(EmployeeID int, EmployeeName varchar(10))
		
		INSERT @TargetTable(EmployeeID, EmployeeName) VALUES(100, 'Mary');
		INSERT @TargetTable(EmployeeID, EmployeeName) VALUES(101, 'Sara');
		INSERT @TargetTable(EmployeeID, EmployeeName) VALUES(102, 'Stefano');
		INSERT @TargetTable(EmployeeID, EmployeeName) Values(103, 'Bob1');
		
		INSERT @SourceTable(EmployeeID, EmployeeName) Values(103, 'Bob');
		INSERT @SourceTable(EmployeeID, EmployeeName) Values(104, 'Steve');
		
		BEGIN TRAN;
		MERGE @TargetTable AS T
		USING @SourceTable AS S
		ON (T.EmployeeID = S.EmployeeID)
		WHEN NOT MATCHED BY TARGET
		   THEN INSERT(EmployeeID, EmployeeName) VALUES(S.EmployeeID, S.EmployeeName)
		WHEN MATCHED
		   THEN UPDATE SET T.EmployeeName = S.EmployeeName
		WHEN NOT MATCHED BY SOURCE
		   THEN DELETE
		OUTPUT $action, inserted.*, deleted.*;
		ROLLBACK TRAN;		

OUTPUT $action, deleted.*,inserted.*;	$action specify the action done in the execution of merge statement in each row. inserted means what will be the data after operation,deleted menas what was the original data.			
Cursor	cursor is a database objects to retrieve data from a result set one row at a time, instead of the T-SQL commands that operate on all the rows in the result set at one time. We use cursor when we need to update records in a database table in singleton fashion means row by row.			
Life Cycle of Cursor :				

Concepts	Discription	Example		
01.Declare Cursor	DECLARE CURSOR defines the attributes of a Transact-SQL server cursor, such as its scrolling behavior and the query used to build the result set on which the cursor operates.	DECLARE cursor_name CURSOR02. [LOCAL | GLOBAL] --define cursor scope03. [FORWARD_ONLY | SCROLL] --define cursor movements (forward/backward)04. [STATIC | KEYSET | DYNAMIC | FAST_FORWARD] --basic type of cursor05. [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC] --define locks 06. FOR select_statement --define SQL Select statement 07. FOR UPDATE [col1,col2,...coln] --define columns that need to be updated	Local :Specifies that the scope of the cursor is local to the batch, stored procedure, or trigger in which the cursor was created. The cursor name is only valid within this scope. The cursor can be referenced by local cursor variables in the batch, stored procedure, or trigger, or a stored procedure OUTPUT parameter. GLOBAL: Specifies that the scope of the cursor is global to the connection. The cursor name can be referenced in any stored procedure or batch executed by the connection. The cursor is only implicitly deallocated at disconnect. FORWARD_ONLY :Specifies that the cursor can only be scrolled from the first to the last row. FETCH NEXT is the only supported fetch option. If FORWARD_ONLY is specified without the STATIC, KEYSET, or DYNAMIC keywords, the cursor operates as a DYNAMIC cursor. When neither FORWARD_ONLY nor SCROLL is specified, FORWARD_ONLY is the default, unless the keywords STATIC, KEYSET, or DYNAMIC are specified. STATIC, KEYSET, and DYNAMIC cursors default to SCROLL. SCROLL :Specifies that all fetch options (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) are available. If SCROLL is not specified in an ISO DECLARE CURSOR, NEXT is the only fetch option supported. SCROLL cannot be specified if FAST_FORWARD is also specified.	
02.Open	OPEN statement populates the result set,	OPEN [GLOBAL] cursor_name --by default it is local		
03.Fetch	FETCH returns a row from the result set	FETCH [NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n]02. FROM [GLOBAL] cursor_name 03. INTO @Variable_name[1,2,..n]		
04.Close	CLOSE statement releases the current result set associated with the cursor	CLOSE cursor_name --after closing it can be reopen		
05.Deallocate	The DEALLOCATE statement releases the resources used by the cursor.	DEALLOCATE cursor_name --after deallocation it can't be reopen		
Types of Cursor : 1.Static	The cursor does not reflect any changes made in the database that affect either the membership of the result set or changes to the values in the columns of the rows that make up the result set. A static cursor does not display new rows inserted in the database after the cursor was opened, even if they match the search conditions of the cursor SELECT statement. If rows making up the result set are updated by other users, the new data values are not displayed in the static cursor. The static cursor displays rows deleted from the database after the cursor was opened. No UPDATE, INSERT, or DELETE operations are reflected in a static cursor (unless the cursor is closed and reopened), By default static cursors are scrollable. SQL Server static cursors are always read-only	if we are updating the data with the cursor,Will changes be updated in base table ?		
2.Keyset	The membership and order of rows in a keyset-driven cursor are fixed when the cursor is opened. Keyset-driven cursors are controlled by a set of unique identifiers, keys, known as the keyset. The keys are built from a set of columns that uniquely identify the rows in the result set. The keyset is the set of the key values from all the rows that qualified for the SELECT statement at the time the cursor was opened. The keyset for a keyset-driven cursor is built in tempdb when the cursor is opened. Changes to data values in nonkeyset columns (made by the cursor owner or committed by other users) are visible as the user scrolls through the cursor. Inserts to the database made outside the cursor are not visible in the cursor unless the cursor is closed and reopened. Inserts made through the cursor using an API function such as the ODBC SQLSetPos function are visible at the end of the cursor. @@FETCH_STATUS returns a "row missing" status when an attempt is made to fetch a row deleted after the cursor was opened. An update to a key column operates like a delete of the old key value followed by an insert of the new key value. The new key value is not visible if the update was not made through the cursor; it is visible at the end of the cursor if the update was made through the cursor using either an API function such as SQLSetPos or the Transact-SQL WHERE CURRENT OF clause and the SELECT statement did not contain a JOIN condition in the FROM clause. The new key value is not visible if the insert contained a remote table in the FROM clause. Attempts to retrieve the old key value get the same missing row fetch status as a deleted row.			
3.Dynamic	A dynamic cursor allows you to see the data updation, deletion and insertion in the data source while the cursor is open. Hence a dynamic cursor is sensitive to any changes to the data source and supports update, delete operations. By default dynamic cursors are scrollable.			
4.Forward Only Cursors	A forward only cursor is the fastest cursor among the all cursors but it doesn't support backward scrolling. You can update, delete data using Forward Only cursor. It is sensitive to any changes to the original data source. There are three more types of Forward Only Cursors.Forward_Only KEYSET, FORWARD_ONLY STATIC and FAST_FORWARD. A FORWARD_ONLY STATIC Cursor is populated at the time of creation and cached the data to the cursor lifetime. It is not sensitive to any changes to the data source. A FAST_FORWARD Cursor is the fastest cursor and it is not sensitive to any changes to the data source.			
READ_ONLY	Prevents updates made through this cursor. The cursor cannot be referenced in a WHERE CURRENT OF clause in an UPDATE or DELETE statement. This option overrides the default capability of a cursor to be updated.	Update Employee SET name='nImesh' WHERE CURRENT OF cursoor		
Update delete and insert in View	http://www.informit.com/articles/article.aspx?p=130855&seqNum=4			
SQL Server 2008 New features	http://sqlhints.com/2011/09/11/new-features-in-sql-server-2008/			
New Date Time Datatype	New types introduced are DATE,TIME,DATETIME2(n), DATETIMEOFFSET			
Five First normalized Form	SYSDATETIME, SYSDATETIMEOFFSET, SYSUTCDATETIME SWITCHOFFSET and TODATETIMEOFFSET	The SYSDATETIME function returns the present system timestamp without the time zone, with an accuracy of 10 milliseconds. The SYSDATETIMEOFFSET function works like SYSDATETIME but the only difference is it includes the time zone. SYSUTCDATETIME returns the Universal Coordinated Time that is known as Greenwich Mean Time within an accuracy of 10 milliseconds. SWITCHOFFSET returns a datetimeoffset value that is changed from the stored time zone offset to a specified new time zone offset.		
	Large UDTs in SQL server 2008 ?			
	Sparse columns	http://sqlhints.com/2014/10/06/a-z-of-filtered-indexes-with-examples-in-sql-server/		
	New HierarchyID datatype	??		
GROUPING SETS, CUBE, and ROLLUP Subclauses	https://technet.microsoft.com/en-us/library/cc721270(v=sql.100).aspx			
Sql Server 2012 features				
column store indexes	When you create a column store index it stores same column data in the same page.	How we know that we required only set of columns only.? No clear understanding.	http://www.codeproject.com/Articles/526621/Top-exciting-features-of-SQL-Server-Part	
Sequence objects		create sequence MySeq as int start with 1 -- Start with value 1 increment by 1-- Increment with value 1 minvalue 0 -- Minimum value to start is zero maxvalue 100 -- Maximum it can go to 100 no cycle -- Do not go above 100 cache 50 -- Increment 50 values in memory rather than incrementing from IO SELECT NEXT VALUE FOR dbo.MySequence AS seq_no;	https://msdn.microsoft.com/en-IN/library/ff878091.aspx	
Pagination	Fetch,Offset			
Contained database	we can have a database with meta-data information, security information etc with in the database itself. So that when we migrate the database, we migrate everything with it			
Error handling				
Metadata discovery improvements				
Projection redirection and the WITH RESULT SETS argument				
database management system principle:	ACID (atomicity, consistency, isolation, and durability)			
Sql Server 2014 features	In-Memory : By moving select tables and stored procedures into memory, you can drastically reduce I/O and improve performance of your OLTP applications.			

Concepts	Discription	Example
	AlwaysOn Availability Groups :SQL Server 2014's AlwaysOn Availability Groups has been enhanced with support for additional secondary replicas and Windows Azure integration.First introduced with SQL Server 2012, AlwaysOn Availability Groups boosted SQL Server availability by providing the ability to protect multiple databases with up to four secondary replicas. In SQL Server 2014, Microsoft has enhanced AlwaysOn integration by expanding the maximum number of secondary replicas from four to eight. Readable secondary replicas are now available for read-only workloads, even when the primary replica is unavailable.	
	Enhancements to Backups:	

PIVOT	PIVOT rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output, and performs aggregations where they are required on any remaining column values that are wanted in the final output.	SELECT <non-pivoted column>,
		   [first pivoted column] AS <column name>,
		   [second pivoted column] AS <column name>,
		   ...
		   [last pivoted column] AS <column name>
		FROM
		   (<SELECT query that produces the data>)
		   AS <alias for the source query>
		PIVOT
		(
		   <aggregation function>(<column being aggregated>)
		FOR
		[<column that contains the values that will become column headers>]
		   IN ([first pivoted column], [second pivoted column],
		   ... [last pivoted column])
		) AS <alias for the pivot table>
		<optional ORDER BY clause>;

WITH CTE() AS	the common table expression (CTE) is a temporary named result set that you can reference within a subsequent SELECT, INSERT, UPDATE, or DELETE statement	
Contained database		https://www.sqlshack.com/contained-databases-in-sql-server/
Offset/Fetch clause		https://www.geeksforgeeks.org/sql-offset-fetch-clause/
What cluster index is		
Why cluster index		https://www.geeksforgeeks.org/sql-indexes/
only one cluster index per table	Only one per table is allowed	
cluster index on not primary key		https://www.c-sharpcorner.com/UploadFile/ff2f08/create-cluster-index-other-than-primary-key-column-in-sql-se/
how cluster index works internally		https://www.red-gate.com/simple-talk/sql/learn-sql-server/effective-clustered-indexes/

Decorator Pattern http://www.journaldev.com/1540/decorator-pattern-in-java-example-tutorial

Design Principles

1

2

3

4 Strive for loosely coupled designs betwwen the objects that interacts

5 Classes should be open for extension, but closed for modification.

Concepts	Discription	Example
ADO.NET		
Data provider	responsible for providing and maintaining the connection to the database. It is a set of related components that work together to provide data in an efficient and performance driven manner	The .NET Data Provider currently comes with two Data Providers - The SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDbDataProvider which allows us to connect to other types of databases like Access and Oracle.
Connection string contains	Data source,attachdbFilename,Integrated Security,server, database,user id,Password,Initial Catalog	<add name="SALES" connectionString="Server=IGTEHYDZSDB01;Database=SEPDB_Online; User Id=int_test;password=int_test" />
Creating connection in using block.	System will dispose the conenction object once the execution will be over in that block	
to get the connection string from config file:	ConfigurationManager.ConnectionStrings[name];	
Connection pooling	http://msdn.microsoft.com/en-us/library/8xx3tyca(v=vs.110).as	**???**
Execute reader	Used to examine the result of query.when u move to next row,Previous row will be discarded.It is read only and forward only.	
command.ExecuteReader()		
command.ExecuteNonQuery()		
command.ExecuteDataSet()		
DataSet is transmitted as XML		
type and untype dataset	http://msdn.microsoft.com/en-us/library/wha85tzb(v=vs.110).aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-3 Along with late bound access to values through weakly typed variables, the DataSet provides access to data through a strongly typed metaphor. Tables and columns that are part of the DataSet can be accessed using user-friendly names and strongly typed variables.	

Transaction	To perform the set of action as a single block.	Sqlconnection con = new sqlConnection();
		Transaction trans = con.beginTransaction();
		SqlCommand cmd = new SqlCommand();
		cmd.Transaction = trans
		.
		.
		.
		Trans.commit()
		.
		.
		Trans.RollBack()

	CultureInfo (InvariantCulture)	.
DbProviderFactories	http://msdn.microsoft.com/en-us/library/wda6c36e(v=vs.110).aspx	
Class		

DbProviderFactory		//get the information out of the configuration file.
		ConnectionStringSettings connectionStringSettings =
		ConfigurationManager.ConnectionStrings["blah"];
		
		//get the proper factory
		DbProviderFactory factory =
		DbProviderFactories.GetFactory(connectionStringSettings.ProviderName);
		
		//create a command of the proper type.
		DbConnection conn = factory.CreateConnection();
		//set the connection string
		conn.ConnectionString = connectionStringSettings.ConnectionString;
		
		//open the connection
		conn.Open();

SqlConnection,DbConnection		
SqlCommand,DbCommand		
SqlParameter,DbParameter		
SqlDataReader		
Dataset in detail		

Dataset Methos and property	WriteXmlSchema()
	WriteXml()
	ReadXml() to read schema and data from XML format.
	AcceptChanges() and RejectChanges()
	EnforceConstraints property=> Constraints are enforced when the EnforceConstraints property of the DataSet is TRUE.	

datatable in detail		
DataColumn		

DataRow	RowState : to track the changes that have been made to a DataRow	To get the original version of deleted row :
		if (dataRow.RowState == DataRowState.Deleted)
		id = (string)dataRow["CustomerID",DataRowVersion.Original];

DataAdapter in detail	Needs to have demo.	
Some properties and methods of DataAdapter	ContinueUpdateOnError	
One practical example of generalized data access ir respective of data provider.		
UniqueKeyConstraint		

Concepts	Discription	Example
ForeignKeyConstraint	The DeleteRule and UpdateRule properties of the ForeignKeyConstraint define the action to be taken when the user attempts to delete or update a row in a related table. The settings for DeleteRule and UpdateRule properties for the ForeignKeyConstraint are as follows:- • Cascade - Deletes or updates related rows. • SetNull - Set values in related rows to DBNull • SetDefault - Set values in related rows to the default value. • None -Take no action on related rows. This is the default behavior	
	SAVE TRANSACTION	

Doubts + points	Solution	Example
To include the jquery files		
$ keyword + why to use		
Start point	$(Documnet).ready()	$(Documnet).ready(Function () { alert("ready function") })
Ready function will be loaded when ? Page life cycle realted		
Other function as a starting point ?		

Use of "ON" keyword	it used to bind event with jquery	$('#id').on('click',function() {
		alert(event Binded);
		});

Types of event in jquery	mouseover,mouseout,click	
Diffenrt functions	fadeIn(),fadeOut().SlideToggle(),toggle()	
this keyword		
how to get attibutevalue	$('#ID').attr('href');	
Callback in Jquery	wait untill the current function completes its execution.	
addClass		
removeClass		
setInterval()		
clearInterval()		
animate()	An object of CSS properties and values that the animation will move toward	
$.ajax({})		
$.each()		
jQuery-1.4.1-vsdoc.js	Provide intelligence to visual studio	
jQuery-1.4.1.js	it is debug version of jquery.	
jQuery-1.4.1.min.js	it is minified versoin of jquery file.	
$()	it is a selector fuction..to select the elements from web page.	
jQuery() and $()	called as selectors.	
Difference between bind(),		http://www.codeproject.com/Articles/662949/Differences-Among-Bind-Live-Delegate-Trigger-in-jQ
:first,:last,:even,:odd,:not,:first-child,:last-child,nth-child()		$("h2:first")
search elements with specified value of attribute	*,^,$,!	$(a[href=mydomain])
:enabled,:disabled		
$(:checked),$(:selected),		
$("h2").size(),$("h2").length()		
$("h2").eq(1)		
$("h2").eq(1).css("color","green")		
children(),find(),	searching in Dom in down direction	This method only traverse a single level down the DOM tree.
parent(),parents(),parentsUntil(),closet() searching in Dom in up direction	closest() : Begins with the current element Travels up the DOM tree and returns the first (single) ancestor that matches the passed expression The returned jQuery object contains zero or one element parents() : Begins with the parent element Travels up the DOM tree and returns all ancestors that matches the passed expression The returned jQuery object contains zero or more than one element Other related methods: parent() : - returns the direct parent element of the selected element parentsUntil() - returns all ancestor elements between two given arguments	

Doubts + points	Solution	Example

siblings()
next()
nextAll()
nextUntil()
prev()
prevAll()
prevUntil()
	Traversing Sideways in The DOM Tree	The siblings() method returns all sibling elements of the selected element.

manipulating	append(),detach(),$("#id1).after($("#div2")), replacewith()	
animation	delay(),fadeIn(),fadeOut(),fadeTo(),show(),hide(), slideDown(),slideUp(),slideToggle(),toggle(),stop()	
ajax		
type:	http request type,GET or POST	type:POST
contentType	request's multipurpose internet mail extension(NIME) type,The content type used when sending data to the server. Default is: "application/x-www-form-urlencoded"	contentType:"application/json"
dataType:	MIME type of data you want to have returned from the request	dataType:"json"
url:		url: "xyz.asmx/getemployee"
data:		data: "{}"
success:		
error:		
async:	A Boolean value indicating whether the request should be handled asynchronous or not. Default is true	
beforeSend(xhr)	A function to run before the request is sent	

7) Explain bind() vs live() vs delegate() methods	The bind() method will not attach events to those elements which are added after DOM is loaded while live() and delegate() methods attach events to the future elements also.
	-The difference between live() and delegate() methods is live() function will not work in chaining. It will work only on an selector or an element while delegate() method can work in chaining.
		$(document).ready(function(){
		$("#myTable").find("tr").live("click",function(){
		alert($(this).text());
		});
		});
		
		Above code will not work using live() method. But using delegate() method we can accomplish this.
		
		$(document).ready(function(){
		$("#dvContainer")children("table").delegate("tr","click",
		function(){
		alert($(this).text());
		});
		});

What is difference between $(this) and 'this' in jQuery?	his and $(this) references the same element but the difference is that "this" is used in traditional way but when "this" is used with $() then it becomes a jQuery object on which we can use the functions of jQuery.	
param()		

jQuery.holdReady()	By using jQuery.holdReady() function we can hold or release the execution of jQuery's ready event.
	This method should be call before we run ready event.
	To delay the ready event, we have To call
	This function is helpful when we want to load any jQuery plugins before the execution of ready event.
		jQuery.holdReady(true);
		
		-When we want to release the ready event then we have to call
		jQuery.holdReady(false);

Doubts + points	Solution	Example
empty() vs .remove() vs .detach().	empty() method is used to remove all the child elements from matched elements. -.remove() method is used to remove all the matched element. This method will remove all the jQuery data associated with the matched element. -.detach() method is same as .remove() method **except** that the .detach() method doesn't remove jQuery data associated with the matched elements. -.remove() is faster than .empty() or .detach() method.	
Is window.onload is different from document. ready()?	The window.onload() is Java script function and document.ready() is jQuery event which are called when page is loaded. - The difference is that document.ready() is called after the DOM is loaded without waiting for all the contents to get loaded. While window.onload() function waits until the contents of page is loaded. - Suppose there is very large image on a page, at that time window.onload() will wait until that image is loaded totally. - So while using the window.onlaod() function the execution will be slow, but the document.ready() will not wait until the image is loaded	

What is Chaining in jQuery?	Chaining means specifying multiple function and/or selectors to an element.
	Advantage of chaining is that it makes your code simple and simple to manage.
	-The execution becomes faster because the code search for the element only once.
		$(document).ready(function(){
		$('#mydiv').css('color', 'blue');
		$('#mydiv').addClass('myclass');
		$('#mydiv').fadeIn('fast');
		}
		
		By using chaining we can write above code as follows
		
		$(document).ready(function(){
		$('#mydiv').css('color', 'blue').addClass('myclass').fadeIn('fast');
		});

What is difference between prop and attr?	In jQuery both prop() and attr() function is used to set/get the value of specified property of an element. The difference in both the function is that attr() returns the default value of the property while the prop() returns the current value of the property.	<input value="My Value" type="text"/> $('input').prop('value', 'Changed Value'); -.attr('value') will return 'My Value' -.prop('value') will return 'Changed Value'
resize()	The resize() function is called whenever the browser size is changed. This event can be only used with $(window).	$(window).resize(function() { $('#message).text('window is resized to ' + $(window).width() + ‘x’ + $(window).height()); });
On,Off,One	on to bind the event,off to unbind event and one to create an event to be executed one time only	
this keyword	http://learn.jquery.com/javascript-101/this-keyword/	
chaining in jQuery	Chaining allows us to run multiple jQuery methods (on the same element) within a single statement. Execution will be from left to right.	
$.Get()	The jQuery.get(url, [data], [callback], [type]) method loads data from the server using a GET HTTP request. The method returns XMLHttpRequest object.	<script type="text/javascript"> $(function() { $.get("content.html", function(data, textStatus) { alert("Done, with the following status: " + textStatus + ". Here is the response: " + data); }); }); </script>

Doubts + points	Solution	Example
	Here is the description of all the parameters used by this method: url: A string containing the URL to which the request is sent data:: This optional parameter represents key/value pairs that will be sent to the server. callback:: This optional parameter represents a function to be executed whenever the data is loaded successfully. type:: This optional parameter represents type of data to be returned to callback function: "xml", "html", "script", "json", "jsonp", or "text".	The first callback parameter is simply the content of the page requested, while the second callback parameter is the textual status of the request

$.Post()		<script type="text/javascript">
		$(function()
		{
		$.post("test_post.php",
		{
		name: "John Doe",
		age: "42"
		},
		function(data, textStatus)
		{
		alert("Response from server: " + data);
		});
		});
		</script>

$("div.a")		
$("div>a")		
Load	The load() method loads data from a server and puts the returned data into the selected element.	$(selector).load(URL,data,callback);
JSON	JSON: JavaScript Object Notation. JSON is a syntax for storing and exchanging data. JSON is an easier to use alternative to XML.	
JSON.parse()	The JavaScript function JSON.parse(text) can be used to convert a JSON text into a JavaScript object	
SOAP	SOAP stands for Simple Object Access Protocol SOAP is a communication protocol SOAP is for communication between applications SOAP is a format for sending messages SOAP communicates via Internet SOAP is platform independent SOAP is language independent SOAP is based on XML	
Namespaced events		$('li').on('click.logging', function() { console.log('a list item was clicked'); });
Unbind the event		$('li').off('click');
preventDefault		
Difference between Empty and remove		
Use of "strict"		

Concepts	Discription	Example
Web service - Defination	A Web Service exposes a number of methods to provide functions that can be used by one or more applications, regardless of the programming languages,operating systems, and hardware platforms used to develop them. The methods that provide such function are called Web Methods.	
	The functions used by a Web Service can be accessed by applications using Internet standards, such as Simple Object Access Protocol (SOAP).	SOAP is a protocol that uses eXtensible Markup Language (XML) to describe data and HyperText Transfer Protocol (HTTP) to transmit application data.
Web service client	An application that uses a Web service is called a Web service client.	
Service Provider Service Broker Service Client	• Service Provider: A service provider is responsible for providing the software components that are published as Web services. The software components can be simple classes or complex applications written in some programming language. A service provider describes information of the Web Service using an interface. The interface specifies information, such as: 1.Service methods that are invoked by a Client 2.URL that the client needs to use to access the service. 3.Network protocol required to access the Web Service.	• Service Broker: The interface defined by the service provider is published in a centralized service registry called a service broker. The service broker allows the Web Clients to search the registry for information about published Web Services. The client and the Web service locate each other using service broker. • Service Client: A service client is the potential client of the service provider's Web Service, whose information is made available by the service broker. A service client, after locating the Web Service in the service registry, invokes the services implemented by the Web Service. Locating a Web Service in the service registry and invoking its methods is known as the Web Service binding operation. A service client can be a simple Web application or another Web service accessing the published Web Services.
Web Service Description Language (WSDL)	Provides format to represent the webs ervice exposed by services provider. WSDl is xml based grammer that describes how external clients can intract with the web methods as given URL,using each of the supported wire protocols WSDL provides follow characteristics 1.The name of the XML web methis 2.number of,type id,and order of input parameter 3.return type 4.the HTTP Get, HTTP POST and SOAP calling conventions	check one example
	To view wsdl for a webservice, type URL for the Web Service followed by ?wsdl as shown below: http://localhost/WebService/Service1.asmx?wsdlPage	
Universal Description, Discovery and Integration (UDDI):	It provides a standard mechanism to register and discover a Web Service. When a web service provider wants to make a web service available to client applications, the provider describes the Web Service by using a WSDL document. Then, the provider registers the Web Service in the UDDI Directory, which contains pointers to the Web Service and the WSDL document for the Web Service.	http://uddi.microsoft.org/
	1. HTTP GET Operation: You can pass parameters to a Web Service by calling the ASMX page with query string parameters for the method to call and the values of simple parameters to pass. Example: WebDemo.asmx/MethodName?Parm1=value 2. HTTP POST Operation: It works the same as GET Operation except that the parameters are passed as standard URL encoded form variables. If you use a client such as wwIPStuff you can use AddPostKey() to add each parameter in the proper parameter order. Example: WebDemo.asmx/MethodName 3. SOAP: This is the proper way to call a Web Service in .Net and it is also the way that .Net uses internally to call Web Services. The GET and POST operations are useful if you need to call a Web Service quickly and no SOAP client is readily available. For example: In a browser based client application it may be easier to use GET and POST instead of constructing and parsing the more complex SOAP headers that are passed back and forth in a SOAP request. However, with a proper SOAP client in place SOAP provides the full flexibility of the protocol, where GET and POST operations have to stick to simple inputs and outputs. Among other things that you can do with SOAP is pass complex objects and data over the wire and for these operations to work you need to use SOAP.	
	Class implementting the wer services method implements the 'System. Web.Services.WebServices'	
WebService and WebMethod attribue		
WebService attribue 's Property		
WebMethod attribue 's Property		

Concepts	Discription	Example	
	service oriented technology		
Advantage	1.WCF is interoperable with other services when compared to .Net Remoting,where the client and service have to be .Net. 2.WCF services provide better reliability and security in compared to ASMX web services. 3.In WCF, there is no need to make much change in code for implementing the security model and changing the binding. Small changes in the configuration will make your requirements. 4.WCF has integrated logging mechanism, changing the configuration file settings will provide this functionality. In other technology developer has to write the code.		
Difference between WCF and Web service		http://www.wcftutorial.net/Difference-between-WCF-and-Webservice.aspx	
what are the alternatives to WCF earlier ?			
Why WCF ?			
WCF Design Goals :	Interoprability Service oriented development Unification of existing distributed architechture		
EndPoint	WCF Service is a program that exposes a collection of Endpoints. Each Endpoint is a portal for communicating with the world. End point consists of three components		
1.Address	Basically URL, specifies where this WCF service is hosted .Client will use this url to connect to the service		
2.Binding	Binding will describes how client will communicate with service. There are different protocols available for the WCF to communicate to the Client. You can mention the protocol type based on your requirements.	Transport -Defines the base protocol to be used like HTTP, Named Pipes, TCP, and MSMQ are some type of protocols. Encoding (Optional) - Three types of encoding are available-Text, Binary, or Message Transmission Optimization Mechanism (MTOM). MTOM is an interoperable message format that allows the effective transmission of attachments or large messages (greater than 64K). Protocol(Optional) - Defines information to be used in the binding such as Security, transaction or reliable messaging capability	
Bindings	BasicHttpBinding,NetTcpBinding		
3.Contract	Collection of operation that specifies what the endpoint will communicate with outside world. Usually name of the Interface will be mentioned in the Contract, so the client application will be aware of the operations which are exposed to the client.	<system.serviceModel> <services> <service name="MathService" behaviorConfiguration="MathServiceBehavior"> <endpoint address="http://localhost:8090/MyService/ MathService.svc" contract="IMathService" binding="wsHttpBinding"/> </service> </services> </system.serviceModel>	
	In WCF, all services are exposed as contracts. Contract is a platform-neutral and standard way of describing what the service does. Mainly there are four types of contracts available in WCF 1.Service Contract /Operation Contract 2.Data Contract 3.Message Contract 4.Fault Contract	How internally this attibute is usefull to framework to generate WSDL	
1.Service Contract	Service contracts describe the operation that service can provide. For Eg, a Service provide to know the temperature of the city based on the zip code, this service is called as Service contract. It will be created using Service and Operational Contract attribute.	System.ServiceModel.ServiceContratAttribute // what all are the attributes on Service Contract CallbackContract= "" ,ConfigurationName="" ,Name="" ,Namespace = "" ,ProtectionLevel="" ,SessionMode = ""	
Operation Contract		System.ServiceModel.OperationContratAttribute Action = "" ,AsyncPattern= false ,IsInitiating = false ,IsOneWay = "" ,IsTerminating = false ,Name = "" ,ProtectionLevel = System.Net.Security.AuthenticationLevel ,ReplyAction = ""	

Concepts	Discription	Example	

2.Data Contract	Data contract describes the custom data type which is exposed to the client. This defines the data types, that are passed to and from service. Data types like int, string are identified by the client because it is already mention in XML schema definition language document, but custom created class or data types cannot be identified by the client e.g. Employee data type. By using DataContract we can make client to be aware of Employee data type that are returning or passing parameter to the method.
	
	Data contract using [DataContract] and [DataMember] attribute
		System.Runtime.Serialization.DataContract
		http://www.wcftutorial.net/Data-Contract.aspx
		Isrederence = ""
		Name = ""
		Namespace = ""	

DataMember		EmitDefault = ""
		IsRequired = ""
		Name= ""
		Order = ""	

3.Message Contract	As I said earlier, WCF uses SOAP message for communication. Most of the time developer will concentrate more on developing the DataContract, Serializing the data, etc. WCF will automatically take care of message. On Some critical issue, developer will also require control over the SOAP message format. In that case WCF provides Message Contract to customize the message as per requirement.	IsWrapped = true
		WrapperName = ""
		WrapperNamespace = ""
		ProtectionLevel=""	

You have to follow certain rules while working with Message contract	When using Message contract type as parameter, Only one parameter can be used in servicie Operation [OperationContract] void SaveEmployeeDetails(EmployeeDetails emp); Service operation either should return Messagecontract type or it should not return any value [OperationContract] EmployeeDetails GetEmployeeDetails(); Service operation will accept and return only message contract type. Other data types are not allowed. [OperationContract] EmployeeDetails ModifyEmployeeDetails(EmployeeDetails emp); Note: If a type has both Message and Data contract, service operation will accept only message contract.		

MessageHeader		Actor = ""
		MustUnderstand = ""
		Name = ""
		Namespace = ""
		Order = ""
		,ProtectionLevel=""
		relay=""	

MessageBodyMember		Name = ""
		Namespace = ""
		Order = ""
		,ProtectionLevel=""	

MessageHeaderArray	Lets say we have
	[MessageContract]
	public class Department
	{
	[MessageHeader]
	public string DepartmentName;
	[MessageHeader]
	public Employees Employee();
	
	}
	Message format will be :
	<Department>
	<DepartmentName>Production</DepartmentName>
	<Employees>
	<Employee>Sam</Employee>
	<Employee>Ram</Employee>
	<Employee>Raja</Employee>
	</Employees>
	</Department>
		If we use MessageHeaderArray in place of MessageHeader for the Employees,Message format will be like below format :
		<Department>
		<DepartmentID>PRO1243</DepartmentID>
		<DepartmentName>Production</DepartmentName>
		<Employee>Sam</Employee>
		<Employee>Ram</Employee>
		<Employee>Raja</Employee>
		</Department>	

,MessageParameter,MessageProperty			
4.Fault Contract	Suppose the service I consumed is not working in the client application. I want to know the real cause of the problem. How I can know the error? For this we are having Fault Contract. Fault Contract provides documented view for error occurred in the service to client. This helps us to easy identity, what error has occurred.		

Concepts	Discription	Example	
wcf hOSTING BY 4 WAYS :	1.Selfhosting 2.IIS 3.Windows Activation Services 4.Windows Services		
1.Self hosting :	Prons : Cons	ServiceHost sh = new Servicehost(srvice) sh.open()	
2.IIS	Prons : Cons		
3.Windows Activation Servers	Prons : Cons		
4.Windows Services	Prons : Cons		
in-process hosting, or in-proc for short	A special case of hosting is in-process hosting, or in-proc for short, where the service resides in the same process as the client		
massage Exchange Pattern :			
	1. OneWay/Simplex 2. Request-Reply 3. Duplex		
OneWay/Simplex			
Request-Reply			
Duplex		Example ??	
Service Behaviour	Defination : ?? ServiceBehavior attribute is used to define service behaviour or in web. config file we can define can we define at service level only ?? Not at operation level ?? Where can we define ?? In Interface or service implementation. Who will override whose effect ?	Attributes ?	
Instance Management (It is one of service behaviour)	Instance management is set of techniques WCF uses to bind client request to service instance, governing which service instance handles which client request.it will finallize whether to create a new instance of service object or need to use the existing.		
1.Per-Call Service :	When WCF service is configured for Per-Call instance mode, Service instance will be created for each client request. This Service instance will be disposed after response is sent back to client.	[ServiceBehavior (InstanceContextMode=InstanceContextMode.PerCall)] How to do in web.config file ??	
2.Per-Session Service:	When WCF service is configured for Per-Session instance mode, logical session between client and service will be maintained. When the client creates new proxy to particular service instance, a dedicated service instance will be provided to the client. It is independent of all other instance. How it will identify that it is first time client is calling (Identify the user ?)... how it will dospose the service instance.	[ServiceBehavior (InstanceContextMode=InstanceContextMode.PerSession)]	
3.Singleton Service :	When WCF service is configured for Singleton instance mode, all clients are independently connected to the same single instance. This singleton instance will be created when service is hosted and, it is disposed when host shuts down.	[ServiceBehavior (InstanceContextMode=InstanceContextMode.Single)]	

Throttling (It is also one of service behaviour)	WCF throttling provides some properties that you can use to limit how many instances or sessions are created at the application level. Performance of the WCF service can be improved by creating proper instance.	In Web.config file :
		<system.serviceModel>
		  <services >
		   <service behaviorConfiguration="ServiceBehavior" name="MyService">
		    </service>
		  </service>
		 <behaviors>
		   <serviceBehaviors>
		    <behavior name="ServiceBehavior">
		     <serviceMetadata httpGetEnabled="true"/>
		    <serviceThrottling maxConcurrentCalls="500" maxConcurrentInstances ="100" maxConcurrentSessions ="200"/>
		    </behavior>
		   </serviceBehaviors>
		  </behaviors>
		 </system.serviceModel>
		Programatically :
		ServiceHost host = new ServiceHost(typeof(MyService));
		      ServiceThrottlingBehavior throttle
		 = host.Description.Behaviors.Find();
		       if (throttle == null)
		       {
		         throttle = new ServiceThrottlingBehavior();
		         throttle.MaxConcurrentCalls = 500;
		         throttle.MaxConcurrentSessions = 200;
		         throttle.MaxConcurrentInstances = 100;
		         host.Description.Behaviors.Add(throttle);
		       }
		
		       host.Open();	

1.maxConcurrentCalls:	Limits the total number of calls that can currently be in progress across all service instances. The default is 16.		
2.maxConcurrentInstances	The number of InstanceContext objects that execute at one time across a ServiceHost. The default is Int32.MaxValue.		
3.maxConcurrentSessions:	A positive integer that limits the number of sessions a ServiceHost object can accept. The default is 10.		
ClientProxy / Client	When we add reference we name it folllowed by clientProxy and while creating the object of the proxy class,We name it as client automatically.		
What is proxy ?			

Concepts	Discription	Example	
How can we generate Proxy	WCF provides : by adding Service reference,It creates proxy for us. SvcUtil: command-line utility to import the service metadata and generate a proxy		
Use of serviceBehaviour attribute	Used to specify theInstanceContexMode for WCF Service class		
Difference between Web Service in ASP.NET & WCF Service	Web Services can be accessed only over HTTP & it works in stateless environment, where WCF is flexible because its services can be hosted in different types of applications. Common scenarios for hosting WCF services are IIS,WAS, Self-hosting, Managed Windows Service. The major difference is that Web Services Use XmlSerializer. But WCF Uses DataContractSerializer which is better in Performance as compared to XmlSerializer.		
Types of WCF services ?			
How to host an WCF services ?			
throttling			
Diifeerent type of bindings	http://www.dotnet-tricks.com/Tutorial/wcf/VE8a200713-Understanding-various-types-of-WCF-bindings.html		
Structure of Configuration of WCF services			
Channels in WCF			
Message Encoding :	binaryMessageEncoding mtomMessageEncoding textMessageEncoding		
Security :	Authentication : Who are you ? authorization : What authenticated person can do ? Message Intigrity : ensures that message hasnot been tempered. (changing the data..not reading the data) Message Confidential : data remain shoud be private --- Encryption (reading the data..not intrested in changing the data)	WS- * Specification ??	
	Securing your WCF service requires knowledge of the WCF security features related to auditing and logging, authentication, authorization, confidentiality, and integrity Bindings and behaviors allow you to configure transfer security, authentication, authorization, impersonation, and delegation as well as auditing and logging.		
Transfer Security	Transfer security is the means by which WCF secures messages over the network WCF gives you two options to implement transfer security: transport security and message security. Transport security secures the entire communication channel (e.g., by using SSL), while message security secures each message individually which security will be provided by Binding and which secuirity will be provided by behaviour		
transport Security	End to End security fine if webservice is on intranet no proxy involved in accessing the service		
Message Security	performance issue bcz every message need to encrypted	<security mode= "Message"> <message clientCredentialtype = "UserName"/> the rest of the secutrity setting for the binding take the default values.	
Security mode	1.none: user will be not be authenticated and No message encryption 2.transport : Transport security model for authenticating the users and protecting the messages. (Default encryption algorthm is Basic 256) 3.Message :Message security model for authenticating the users and protecting the messages. 4.Both : can appyle message level and transport level security		
clientCredentialtype option for Message Security	1.none:no authentication 2.Windows 3.certificate 4.username: 5.IssueToken	How it works ?	
clientCredentialtype option for transport Security	1.None : No authetication 2.basic : User will be authenticated against username and passeword against active directory 2.NTLM :Workgroup Environment 4.Windows: 5.Certificate: x 509 certificate		
Protection level Mode	1.none : Disable the message protection 2.Sign : sign the data but doesn't encrpt the data 3.Encrypt and sign : sign the data and encrpt the data	[ServiceContract(ProtctionLevel = ProtectionLevel.Sign)	
WCF 4.0	DefaulConfiguration : 1.DefaultEndPoint: 2.DefaultBinding 3.DefaultBehaviour: File LessActivation : discovery		

Concepts	Discription	Example	
MSMQ :	Queued transport provides isolation between the sender and receiver so that if either the sender or receiver were to stop functioning or the communication between them breaks down, the other party can continue to function and the delivery of the message is still queued and available for delivery. Windows Communication Foundation provides support for queues by leveraging Microsoft Message Queuing (MSMQ) as a transport. The roll of the queue is to catch any messages sent between sender and receiver, and to send them to their destination.		
Transaction :	(Scenario) Consider for example client calling multiple service or service itself calling another service, this type of system are called as Distributed Service-oriented application. Now the questions arise that which service will begin the transaction? Which service will take responsibility of committing the transaction? How would one service know what the rest of the service feels about the transaction? Service could also be deployed in different machine and site. Any network failure or machine crash also increases the complexity for managing the transaction.	System.Transaction	
Transaction Manager :	Transaction Manager is the third party for the service that will manage the transaction using two phase committed protocol.		

transactionFlow :	We can specify whether or not client transaction is propagated to service by changing Binding and operational contract configuration	`<bindings>`
		`<netTcpBinding>`
		`<binding transactionFlow="true"></binding>`
		`</netTcpBinding>`
		`</bindings>`	

TransactionFlowAttribute:	Even after enabling transaction flow does not mean that the service wants to use the client's transaction in every operation. We need to specify the TransactionFlowAttribute in operational contract to enable transaction flow. 1.NotAllowed : Client cannot propagate its transaction to service even client has transaction 2.Allowed : Service will allow to flow client transaction.It is not necessary that service to use client transaction. 3.Mandatory : Both Service and client must use transaction aware binding	[OperationContract (TransactionScopeRequired = true, TransactionAutoComplete)] [TransactionFlow(TransactionFlowOption.Allowed)] int Add(int a, int b);	
Operation behaviour Attribute :	1.TransactionScopeRequired : 2.TransactionAutoComplete:		
Service behaviour Attribute :	1.TransactionAutoCompleteonSessionClose 2.TransactionTimeOut : 3.transactionUIsolationLevel :		
REST architecture :	Principales :1.Resources 2.UniqueResourceIdentifier 3.Simple Interface 4.Representation 5.stateless		
[WebGet] [Webinvoke] uritemplate -- uritemplatetable			
Creating proxy :			
creating proxy at design time :	By using svcutil.exe tool :we need to give command to this tool (prompt command) ex(command example): svcutil.exe [/t:code] <metadataDocumentPath>* | <url>* | <epr> First it will download the metadata from the address specified in the command This tool create proxy class according the metadata and config file (output. config). proxy will not be having reference to service implimentation but the service contract. Online retrieval follows either the WS-MetadataExchange protocol or the Microsoft Discovery (DISCO) protocol.		

Concepts	Discription	Example	

creating proxy at runtime:	by using channel factory from the address,bindings,and contract details and then call createChannel on the factory	public class Test
		{
		public void Test()
		{
		BasicHttpBinding myBinding = new BasicHttpBinding();
		EndpointAddress myEndpoint = new EndpointAddress ("http://localhost/MyService/");
		ChannelFactory<imyservice> myChannelFactory = new ChannelFactory<imyservice>(myBinding, myEndpoint);
		IMyService pClient = myChannelFactory.CreateChannel();
		pClient.DoSomething();
		((IClientChannel)pClient).Close();
		}
		}	

Instance Deactivation :	RealeaseInstanceMode.None RealeaseInstanceMode.BeforeCall RealeaseInstanceMode.AfterCall RealeaseInstanceMode.BeforeAndAfterCall	[OperationBehavior (ReleaseInstanceMode=ReleaseInstanceMode.BeforeCall]	
Configuration Setting Format: <System.SeviceModel> </System.SeviceModel>	<behaviors> <endpointBehaviors> <behavior name=""> <webHttp helpEnabled="true" /> </behavior> </endpointBehaviors> <serviceBehaviors> <behavior name=""> <serviceMetadata httpGetEnabled="true" /> <dataContractSerializer maxItemsInObjectGraph="2147483647" /> </behavior> </serviceBehaviors> </behaviors>	<bindings> <netTcpBinding> <binding name="" > </binding> </netTcpBinding> </bindings> <services> <service behaviorConfiguration="" name=""> <endpoint address="" behaviorConfiguration="" binding="" bindingConfiguration="" name="" contract="" /> </service > </services>	
	<client> <endpoint address="" behaviorConfiguration="" binding="" bindingConfiguration="" name="" contract="" /> </client> <comContracts> </comContracts> This section can only be defined in the machine.config file. <commonBehaviors> </commonBehaviors>	<diagnostics> </diagnostics> <extensions> </extensions> <protocolMapping> </protocolMapping> <routing> </routing> <serviceHostingEnvironment> </serviceHostingEnvironment> <standardEndpoints> </standardEndpoints>	
Why WCF			
.NET Remoting	.NET Remoting is a mechanism for communicating between objects which are not in the same process..NET objects are exposed to remote processes, thus allowing inter process communication. The applications can be located on the same computer, different computers on the same network, or on computers across separate networks.Microsoft .NET Remoting provides a framework that allows objects to interact with each other across application domains. What are remote objects? Any object outside the application domain of the caller application should be considered remote, where the object will be reconstructed. Local objects that cannot be serialized cannot be passed to a different application domain, and are therefore non remotable.		
MSMQ (Microsoft Message Queuing) :	Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues.		
DCOM / COM:	DCOM (Distributed Component Object Model) is a set of Microsoft concepts and program interfaces in which client program object s can request services from server program objects on other computers in a network. DCOM is based on the Component Object Model (COM), which provides a set of interfaces allowing clients and servers to communicate within the same computer.		
WCF Hosting in IIS	http://csharp-video-tutorials.blogspot.in/2014/02/part-29-hosting-wcf-service-in-iis.html		
Advantage and disadvsntsge :	http://csharp-video-tutorials.blogspot.in/2014/02/part-30-advantages-and-disadvantages-of.html		
Message format in Web service and RESTfull service :	http://tutorials.jenkov.com/web-services/message-formats.html		

Concepts	Discription	Example	
Rest guidelines :	1.Consider everything as Resources 2.URI 3.Keep Interface simple 4.REQUEST Response should be in representive format 5.Be stateless.		
	TO expose the WCF as WCF RESTfull service,framework provides two attribute webGet,webInvoke.		
	WE need to provide the method and uriTemplate.		
	Difference between WCF service and WCF service is how client communicate to server.		
	In Normal WCF service,Client calls the server's method,Request wll be sent to server in soap message format with HTTP protocol		
	Server Exceutes the method and send the reply back to client in SOAP message format.		
	IN RESTfull WCF service,client consider all methos at server as Resources and send the request as to access the server's resources by providing appropriate http method and URL.Message fomrat can be in any format lie XML,JSON.		
WCF Extensibility – Behavior configuration extensions			
Transport and encoding for common bindings	Name Transport Encoding Interoperable		
	BasicHttpBinding HTTP/HTTPS Text, MTOM Yes		
	NetTcpBinding TCP Binary No		
	IPC Binary No		
	NetMsmqBinding MSMQ Binary No		
	WSHttpBinding HTTP/HTTPS Text, MTOM Yes		
SvcConfigEditor			

Concepts	Discription	Example
DOCTYPE declaration for HTML5	The <!DOCTYPE> declaration must be the very first thing in your HTML doc The <!DOCTYPE> declaration is not an HTML tag; it is an instruction to the In HTML 4.01, the <!DOCTYPE> declaration refers to a DTD, because HTML HTML5 is not based on SGML, and therefore does not require a reference	<!DOCTYPE html>
SGML (Standard Generalized Markup Language)	is a standard for how to specify a document markup language or tag set. Such a specification is itself a document type definition (DTD).	
character encoding (charset) declaration	<metacharset="UTF-8">	
The default character encoding in HTML5 is UTF-8.		
<address>	<address> tag defines the contact information for the author/owner of a document or an article.	<address> Written by <a href="mailto:webmaster@example.com">Jon Doe</a>. </address>
<abbr>		The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.
<datalist>		<input list="browsers"> <datalist id="browsers"> <option value="Internet Explorer"> <option value="Firefox"> <option value="Chrome"> <option value="Opera"> <option value="Safari"> </datalist>
<details>		<details> <summary>Copyright 1999-2014.</summary> <p> - by Refsnes Data. All Rights Reserved.</p> <p>All content and graphics on this web site are the property of the company Refsnes Data.</p> </details>
Using the <dialog> element:		<table> <tr> <th>January <dialog open>This is an open dialog window</dialog></th> <th>February</th> <th>March</th> </tr> <tr> <td>31</td> <td>28</td> <td>31</td> </tr> </table>
<meter>		<meter value="2" min="0" max="10">2 out of 10</meter> <meter value="0.6">60%</meter>
<progress>		<progress value="22" max="100"></progress>
<tt>	Define teletype text	<p><tt>Teletype text</tt></p>
<wbr>	The <wbr> (Word Break Opportunity) tag specifies where in a text it would be ok to add a line-break	<p> To learn AJAX, you must be familiar with the XML<wbr>Http<wbr>Request Object. </p>
novalidate Attribute	The novalidate attribute is a <form> attribute When present, novalidate specifies that form data should not be validated when submitted	
autofocus Attribute	The autofocus attribute is a boolean attribute. When present, it specifies that an <input> element should automatically get focus when the page loads.	
formaction Attribute	The formaction attribute specifies the URL of a file that will process the input control when the form is submitted. The formaction attribute overrides the action attribute of the <form> element. The formaction attribute is used with type="submit" and type="image".	
formenctype Attribute	The formenctype attribute specifies how the form-data should be encoded when submitting it to the server (only for forms with method=" post") The formenctype attribute overrides the enctype attribute of the <form> element. The formenctype attribute is used with type="submit" and type="image".	
formmethod Attribute	The formmethod attribute defines the HTTP method for sending form-data to the action URL. The formmethod attribute overrides the method attribute of the <form> element. The formmethod attribute can be used with type="submit" and type=" image".	
formnovalidate Attribute	The novalidate attribute is a boolean attribute. When present, it specifies that the <input> element should not be validated when submitted. The formnovalidate attribute overrides the novalidate attribute of the <form> element. The formnovalidate attribute can be used with type="submit".	
What are “web workers”?	A web worker is a script that runs in the background (i.e., in another thread) without the page needing to wait for it to complete. The user can continue to interact with the page while the web worker runs in the background. Workers utilize thread-like message passing to achieve parallelism.	
8 new semantic HTML elements	header, section, footer, aside, nav, main, article, figure	

	You can add any new element to HTML with a browser trick	<html>
		
		<head>
		<title>Creating an HTML Element</title>
		<script>document.createElement("myHero")</script>
		<style>
		myHero {
		display: block;
		background-color: #ddd;
		padding: 50px;
		font-size: 30px;
		}
		</style>
		</head>
		
		<body>
		
		<h1>My First Heading</h1>
		
		<p>My first paragraph.</p>
		
		<myHero>My First Hero</myHero>
		
		</body>
		</html>

		<article> Defines an article in the document <aside> Defines content aside from the page content <bdi> Defines a part of text that might be formatted in a different direction from other text <details> Defines additional details that the user can view or hide <dialog> Defines a dialog box or window <figcaption> Defines a caption for a <figure> element <figure> Defines self-contained content, like illustrations, diagrams, photos, code listings, etc. <footer> Defines a footer for the document or a section <header> Defines a header for the document or a section <main> Defines the main content of a document <mark> Defines marked or highlighted text <menuitem> Defines a command/menu item that the user can invoke from a popup menu <meter> Defines a scalar measurement within a known range (a gauge) <nav> Defines navigation links in the document <progress> Defines the progress of a task <rp> Defines what to show in browsers that do not support ruby annotations <rt> Defines an explanation/pronunciation of characters (for East Asian typography) <ruby> Defines a ruby annotation (for East Asian typography) <section> Defines a section in the document <summary> Defines a visible heading for a <details> element <time> Defines a date/time <wbr> Defines a possible line-break
		Difference Between <article> <section> and <div> the <section> element is defined as a block of related elements. The <article> element is defined as a complete, self-contained block of related elements. he <div> element is defined as a block of children elements.
New Form Elements	<datalist> Defines pre-defined options for input controls <keygen> Defines a key-pair generator field (for forms) <output> Defines the result of a calculation	
New Input Types	color date datetime datetime-local email month number range search tel time url week	input type="number" input type="email" input type="url" input type="color" input type="search" input type="date"
New Input Attributes	autocomplete autofocus form formaction formenctype formmethod formnovalidate formtarget height and width list min and max multiple pattern (regexp) placeholder required step	
New Attribute Syntax	Type Example Empty <input type="text" value="John Doe" disabled> Unquoted <input type="text" value=John> Double-quoted <input type="text" value="John Doe"> Single-quoted <input type="text" value='John Doe'>	
HTML5 Graphics	<canvas> Defines graphic drawing using JavaScript <svg> Defines graphic drawing using SVG	
New Media Elements	<audio> Defines sound or music content <embed> Defines containers for external applications (like plug-ins) <source> Defines sources for <video> and <audio> <track> Defines tracks for <video> and <audio> <video> Defines video or movie content	
In the HTML5 standard, the <html> tag, the <body> tag, and the <head> tag can be omitted.		
Custom Attributes:	A custom data attribute starts with data- and would be named based on your requirement.	<div class="example" data-subject="physics" data-level="complex" > ... </div>

		<style>
		myHero {
		    display: block;
		    background-color: #dddddd;
		    padding: 50px;
		    font-size: 30px;
		}
		<body>
		
		<h1>A Heading</h1>
		<myHero>My Hero Element</myHero>
		
		</body>

HTML5 Browser Support	HTML5 defines eight new semantic elements. All these are block-level eler	To secure correct behavior in older browsers, you can set the CSS display property for these HTML elements to block:
		header, section, footer, aside, nav, main, article, figure {
		    display: block;
		}

Canvas	The HTML <canvas> element is used to draw graphics, on the fly, via JavaS
	The <canvas> element is only a container for graphics. You must use JavaS
	Canvas has several methods for drawing paths, boxes, circles, text, and ad
		<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
		</canvas>
		JS:
		var c = document.getElementById("myCanvas");
		var ctx = c.getContext("2d");
		ctx.moveTo(0, 0);
		ctx.lineTo(200, 100);
		ctx.stroke();

SVG	SVG stands for Scalable Vector Graphics SVG is used to define graphics for the Web SVG is a W3C recommendation The HTML <svg> element is a container for SVG graphics. SVG has several methods for drawing paths, boxes, circles, text, and graphic images.	
Differences Between SVG and Canvas	SVG is a language for describing 2D graphics in XML. Canvas draws 2D graphics, on the fly (with a JavaScript). SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element. In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape. Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.	
Comparison of Canvas and SVG		
Canvas	SVG	
Resolution dependent No support for event handlers Poor text rendering capabilities You can save the resulting image as .png or .jpg Well suited for graphic-intensive games	Resolution independent Support for event handlers Best suited for applications with large rendering areas (Google Maps) Slow rendering if complex (anything that uses the DOM a lot will be slow) Not suited for game applications	

Google Maps	Google Maps allows you to display maps on your web page	<!DOCTYPE html>
		<html>
		<body>
		
		<h1>My First Google Map</h1>
		
		<div id="map" style="width:400px;height:400px;background:yellow"></div>
		<script>
		function myMap() {
		var mapOptions = {
		    center: new google.maps.LatLng(51.5, -0.12),
		    zoom: 10,
		    mapTypeId: google.maps.MapTypeId.HYBRID
		}

	Example Explained
	The mapOptions variable defines the properties for the map.
	The center property specifies where to center the map (using latitude and
	The zoom property specifies the zoom level for the map (try to experimen
	The mapTypeId property specifies the map type to display. The following r
	The line: var map=new google.maps.Map(document.getElementById("ma
	e.g
		var map = new google.maps.Map(document.getElementById("map"), mapOptions);
		}
		</script>
		
		<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBu-916DdpKAjTmJNIgngS6HL_kDIKU0aU&callback=myMap"></script>
		<!--
		To use this code on your website, get a free API key from Google.
		Read more at: https://www.w3schools.com/graphics/google_maps_basic.asp
		-->
		
		</body>
		</html>

Videos	Before HTML5, a video could only be played in a browser with a plug-in (lik The HTML5 <video> element specifies a standard way to embed a video in	<video width="320" height="240" controls> <source src="movie.mp4" type="video/mp4"> <source src="movie.ogg" type="video/ogg"> Your browser does not support the video tag. </video>
	The controls attribute adds video controls, like play, pause, and volume. It is a good idea to always include width and height attributes. If height and width are not set, the page might flicker while the video loads. The <source> element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format. The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element.	
<object> Element	The <object> element is supported by all browsers. The <object> element defines an embedded object within an HTML docum It is used to embed plug-ins (like Java applets, PDF readers, Flash Players) i	<object width="100%" height="500px" data="snippet.html"></object> <object data="audi.jpeg"></object> <object width="400" height="50" data="bookmark.swf"></object>
<embed> Element	The <embed> element is supported in all major browsers. The <embed> element also defines an embedded object within an HTML d Web browsers have supported the <embed> element for a long time. How	<embed width="400" height="50" src="bookmark.swf"> <embed width="100%" height="500px" src="snippet.html"> <embed src="audi.jpeg">

Drag and Drop	<!DOCTYPE HTML>
	<html>
	<head>
	<script>
	function allowDrop(ev) {
	    ev.preventDefault();
	}	

	function drag(ev) {
	    ev.dataTransfer.setData("text", ev.target.id);
	}
	
	function drop(ev) {
	    ev.preventDefault();
	    var data = ev.dataTransfer.getData("text");
	    ev.target.appendChild(document.getElementById(data));
	}
	</script>
	</head>
	<body>	

HTML Web Storage	With web storage, web applications can store data locally within the user's
	Before HTML5, application data had to be stored in cookies, included in ev
	Unlike cookies, the storage limit is far larger (at least 5MB) and informatio
	Web storage is per origin (per domain and protocol). All pages, from one o
	
	HTML web storage provides two objects for storing data on the client:
		window.localStorage - stores data with no expiration date
		window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)
		
		e.g.
		localStorage.setItem("lastname", "Smith");
		// Retrieve
		document.getElementById("result").innerHTML = localStorage.getItem("lastname");

The EventSource Object	A server-sent event is when a web page automatically gets updates from a server.	

Concepts	Discription	Example
What is XML?	Extensible Markup Language (XML) is the universal language for data on the Web XML is a technology which allows us to create our own markup language. XML documents are universally accepted as a standard way of representing information in platform and language independent manner. XML is universal standard for information interchange. XML documents can be created in any language and can be used in any language.	
What is a well-formed XML document?	If a document is syntactically correct it can be called as well-formed XML documents.	
A well-formed document conforms to XML's basic r	Every open tag must be closed. The open tag must exactly match the closing tag: XML is case-sensitive. All elements must be embedded within a single root element. Child tags must be closed before parent tags. A well-formed document has correct XML tag syntax, but the elements might be invalid for the specified document type.	
How does the XML structure is defined?	using Document Type Definition (DTD) Schema	
What is DTD?	A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines rules for a specific type of document, including: Names of elements, and how and where they can be used The order of elements Proper nesting and containment of elements Element attributes	

Concepts	Discription	Example
AJAX	AJAX is about updating parts of a web page, without reloading the whole page.	

XMLHttpRequest	The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.	var xmlhttp;
		if (window.XMLHttpRequest)
		  {// code for IE7+, Firefox, Chrome, Opera, Safari
		  xmlhttp=new XMLHttpRequest();
		  }
		else
		  {// code for IE6, IE5
		  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
		  }

open(method,url,async)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not.
	
	method: the type of request: GET or POST
	url: the location of the file on the server
	async: true (asynchronous) or false (synchronous)
		xmlhttp.open("GET","ajax_info.txt",true);
		xmlhttp.onredayStateChange = handler;
		xmlhttp.send();

send(string)	Sends the request off to the server.
	
	string: Only used for POST requests
		xmlhttp.open("GET","ajax_info.txt",true);
		xmlhttp.send();

onreadystatechange	The onreadystatechange event is triggered every time the readyState changes.	var handler = function() {
		}

XMLHttpRequest.responseText	get the response data as a string	
XMLHttpRequest.responseXML	get the response data as XML data	
(xmlhttp.readyState==4 && xmlhttp.status==200)		
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready	
status	200: "OK" 404: Page not found	

Concepts	Discription	Example
AsNoTracking()		
DbConext		
DbSet<>		
AddOrUpdate()		

Concepts	Discription	Example
NodeJS	because it allows to perform non blocking operation. It is actually usuful for Non blocking I/O operations. Node JS is singe threaded. Contradictory : if it is singe threaded then it will give poor performance. But is giving high performance.	
Middleware		
next('route')		
app.get / app.post		
app.use		
app.set		
Prototype, Inheritance in Javascript		
Express JS		
Routing		
Session		
Templating language		
Nginx		
npm		
Non blocking		
Closure		
cwd vs _dirname		
nginx	nginx [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server	

Doubts + points	Solution	Example
What is Type script		
What type scri[pt ad as advantages		

Declaration files	When a TypeScript script gets compiled there is an option to generate a declaration file (with the extension .d.ts) that functions as an interface to the components in the compiled JavaScript. In the process the compiler strips away all function and method bodies and preserves only the signatures of the types that are exported. The resulting declaration file can then be used to describe the exported virtual TypeScript types of a JavaScript library or module when a third-party developer consumes it from TypeScript.	declare module arithmetics {
		    add(left: number, right: number): number;
		    subtract(left: number, right: number): number;
		    multiply(left: number, right: number): number;
		    divide(left: number, right: number): number;
		}

Language features: Type annotations and compile-time type checking Type inference Type erasure Interfaces Enumerated type Mixin Generic Namespaces Tuple Await The following features are backported from ECMAScript 2015: Classes Modules[27] Abbreviated "arrow" syntax for anonymous functions Optional parameters and default parameters		
	TypeScript provides **static typing** through type annotations to enable type checking at compile time.	
	The annotations for the primitive types are **number, boolean and string**. Weakly- or dynamically-typed structures are of type **any**	
Compiler	The TypeScript compiler, named tsc, is written in TypeScript that can be compiled into regular JavaScript that can be executed in any JavaScript engine in any host, such as a browser	
Mixin	In object-oriented programming languages, a mixin is a class that contains methods for use by other classes without having to be the parent class of those other classes. How those other classes gain access to the mixin's methods depends on the language. Mixins are sometimes described as being "included" rather than "inherited".	
Class	Also of note, the use of public on arguments to the constructor is a shorthand that allows us to automatically create properties with that name.	

Doubts + points	Solution	Example
Following are the principles to secure web application		
1. Defense in Depth	Each layer should act as if it is always interacting directly with the outside world, authenticating and authorizing users before allowing them to perform any actions.	
2. Never Trust Input	Any input from a user or another system should always be treated as a potential threat, so always be sure to validate any input before using it. Don't ever assume that you can trust the data because it has already been validated elsewhere.	
3. Enforce the Principle of Least Privilege	For example, instead of running an entire website under an administrator account just to allow disk access to save uploaded files, create a user account that has no direct access to the local machine, except to a specific folder where the account has access to create new files but not to delete, update, or execute them.	
4. Assume External Systems Are Insecure		
5. Reduce Surface Area		ASP.NET MVC's model binding **BindAttribute** provides **Include** and **Exclude** properties that allow you to specify a comma-delimited list of model properties that should be bound or ignored, respectively.
6. Disable Unnecessary Features		
Sql Injection	A SQL injection attack occurs when an attacker tricks the web application ir parameters that cause a query you may not have planned for that uses unt data to be run.	1. Query manipulation to bypass login dredentials 2. Modify data using insert delete updae query. 3. Changing query to generate exception that might give code/databse schema details to user.
	Guidance from Microsoft: SQL injection attacks can be performed in Entity SQL by supplying malicious input to values that are used in a query predicate and in parameter names. To avoid the risk of SQL injection, you should never combine user input with Entity SQL command text. You should use parameterized queries instead of injecting literals from an external agent directly into the query.	
	LINQ to Entities injection attacks: Although query composition is possible in LINQ to Entities, it is performed through the object model API. Unlike Entity SQL queries, LINQ to Entities queries are not composed by using string manipulation or concatenation, and they are not susceptible to traditional SQL injection attacks.	
Cross-Site Scripting		
Cross-Site Request Forgery		
Loging	Authentication	
	Authorization	
Session management		
Server side and client side validation		
Two way data binding		
Caching		
How windows authentication works on Web		
Deployment		
Explore Cross-Site Scripting		
Introduction of IIS and its major features		
What behaviour we inherit by inhering controller class.		
Async and await keyword		
Task		
WPF: Command		
WPF: Static resource and Dynamic resource		
WPF: Sigle thread application		
WPF: Dispacher and dependency		

Doubts + points	Solution	Example
Attached Property	The purpose of dependency properties is to provide a way to compute the value of a property based on the value of other inputs. These other inputs might include system properties such as themes and user preference, just-in-time property determination mechanisms such as data binding and animations/storyboards, multiple-use templates such as resources and styles, or values known through parent-child relationships with other elements in the element tree. In addition, a dependency property can be implemented to provide self-contained validation, default values, callbacks that monitor changes to other properties, and a system that can coerce property values based on potentially runtime information. Derived classes can also change some specific characteristics of an existing property by overriding dependency property metadata, rather than overriding the actual implementation of existing properties or creating new properties.	
	The type that defines the Attached Property is designed so that it can be the parent element of the elements that will set values for the Attached Property. The type then iterates its child objects using internal logic against some object's tree structure, obtains the values and acts on those values in some manner. The type that defines the Attached Property will be used as the child element for a variety of possible parent elements and content models. The type that defines the Attached Property represents a service. Other types set values for the Attached Property. Then, when the element that set the property is evaluated in the context of the service, the Attached Property values are obtained using internal logic of the service class.	
	Attached properties are a type of dependency property. The difference is in how they're used. With an attached property, the property is defined on a class that isn't the same class for which it's being used.	
	One purpose of an attached property is to allow different child elements to specify unique values for a property that is actually defined in a parent element. A specific application of this scenario is having child elements inform the parent element of how they are to be presented in the user interface (UI).	
	In Windows Presentation Foundation (WPF), most of the attached properties that exist on WPF types that are related to UI presentation are implemented as dependency properties. Attached properties are a XAML concept, whereas dependency properties are a WPF concept. Because WPF attached properties are dependency properties, they support dependency property concepts such as property metadata, and default values from that property metadata.	

	public static bool **Get**AllowOnlyString(DependencyObject obj)
	{
	return (bool)obj.GetValue(AllowOnlyStringProperty);
	}
	public static void **Set**AllowOnlyString(DependencyObject obj, bool value)
	{
	obj.SetValue(AllowOnlyStringProperty, value);
	}
	// Using a DependencyProperty as the backing store for AllowOnlyString. This enables animation, styling, binding, etc...
	public static readonly DependencyProperty AllowOnlyStringProperty = DependencyProperty.**RegisterAttached**("AllowOnlyString", typeof(bool),typeof (TextblockExtension), new PropertyMetadata(false, AllowOnlyString));
	private static void AllowOnlyString(DependencyObject d, DependencyPropertyChangedEventArgs e)
	{
	if (d is TextBox)
	{
	TextBox txtObj = (TextBox)d;
	txtObj.TextChanged += (s, arg) =>
	{
	TextBox txt = s as TextBox;
	if (!Regex.IsMatch(txt.Text, "^[a-zA-Z]*$"))
	{
	txtObj.BorderBrush = Brushes.Red;
	MessageBox.Show("Only letter allowed!");
	}
	};
	}
	}	

Dependency Property	public class Button : ButtonBase
	{
	// The dependency property
	public static readonly DependencyProperty IsDefaultProperty;
	static Button()
	{
	// Register the property
	Button.IsDefaultProperty = DependencyProperty.Register(“IsDefault”,
	typeof(bool), typeof(Button),
	new FrameworkPropertyMetadata(false,
	new PropertyChangedCallback(OnIsDefaultChanged)));
	…
	}
	// A .NET property wrapper (optional)
	public bool IsDefault
	{
	get { return (bool)GetValue(Button.IsDefaultProperty); }
	set { SetValue(Button.IsDefaultProperty, value); }
	}
	// A property changed callback (optional)
	private static void OnIsDefaultChanged(
	DependencyObject o, DependencyPropertyChangedEventArgs e) { … }
	…
	}	

	A dependency property depends on multiple providers for determining its value at any point in time.	The motivation for adding such intelligence to properties is to enable rich functionality directly from declarative markup. The key to WPF’s declarative-friendly design is its heavy use of properties
		Change notification . Property value inheritance . Support for multiple providers
		because IsDefaultProperty is a static field (rather than an instance field), the dependency property implementation saves per-instance memory compared to a typical .NET property. Bu How that needs to explore.

Dependency Property metadata	Optionally (via different overloads of Register), you can pass metadata that customizes how the property is treated by WPF, as well as callbacks for handling property value changes, coercing values, and validating values. Button calls an overload of Register in its static constructor to give the dependency property a default value of false and to attach a delegate for change notifications.	
	Local value 2. Parent template trigger 3. Parent template 4. Style triggers 5. Template triggers 6. Style setters 7. Theme style triggers 8. Theme style setters 9. Property value inheritance 10. Default value	
Why dependency property has multiple copy of data even though it is static.	The magic here is, the declaration of DependencyProperty is static not its value (i.e the memory storage). The declaration that you add with static keyword is just the identifier of the DependencyProperty for the DependencyObject because the same identifier will be shared by all the instances of that DependencyObject to identify the property hence it makes sense to make it static. On the other hand, when you set the value of DependancyProperty by calling the SetValue on DependancyObject instance, then each instance of DependancyObject on which the setvalue is called will store its local value of the Property. This is handled internally by the DependancyObject class which maintain sort of Dictionary which has the mapping between the DependancyProperty identifier and the local value.	
Default value will be store in PropertyFromName of DependencyProperty.		
Local value will be store in hasttable format in DependencyObject.		
Difference between Dependency and Attached Properties:	1/ Dependency properties are wrapped in CLR property where as Attached properties are not. 2/ Attached properties and normal dependency properties include the methods used to register them with the property system. For attached properties DependencyProperty.RegisterAttached, .RegisterAttachedReadOnly & for dependency properties .Register and .RegisterReadOnly. 3/ On the basis of Where the property is stored, Attached properties are stored on elements which consume the property and not by the element which exposes the attached property. 4/ Attached properties allows container to create a property which can be used by any child UI elements. Where as dependency property is associated with that particular elements and can help in notification of changes and reacting to that changes	