Here’s a **crash course of C# basics** — simple and clear, just what you need to get started.

## ✅ 1. What is C#?

C# (pronounced "C-sharp") is a modern, object-oriented programming language developed by Microsoft.  
It runs on the **.NET framework** and is widely used for building Windows apps, web apps (with ASP.NET), games (Unity), and more.

## ✅ 2. Basic Structure of a C# Program

csharp

CopyEdit

using System;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello, World!");

}

}

* using System; → imports the System namespace (for basic functions like Console).
* Main → the entry point of the program.
* Console.WriteLine() → prints to the terminal.

## ✅ 3. Variables and Data Types

csharp

CopyEdit

int number = 10;

float pi = 3.14f;

double largePi = 3.14159;

char letter = 'A';

string name = "Vishnu";

bool isAlive = true;

* int, float, double, char, string, bool are common data types.

## ✅ 4. Conditionals

csharp

CopyEdit

int age = 20;

if (age >= 18)

{

Console.WriteLine("Adult");

}

else

{

Console.WriteLine("Minor");

}

## ✅ 5. Loops

### For Loop

csharp

CopyEdit

for (int i = 0; i < 5; i++)

{

Console.WriteLine(i);

}

### While Loop

csharp

CopyEdit

int i = 0;

while (i < 5)

{

Console.WriteLine(i);

i++;

}

## ✅ 6. Methods (Functions)

csharp

CopyEdit

static void Greet(string name)

{

Console.WriteLine("Hello " + name);

}

// Call it like:

Greet("Vishnu");

* void means the function returns nothing.
* You can also return values:

csharp

CopyEdit

static int Add(int a, int b)

{

return a + b;

}

## ✅ 7. Classes and Objects

csharp

CopyEdit

class Person

{

public string Name;

public int Age;

public void Speak()

{

Console.WriteLine($"Hi, I am {Name} and I am {Age} years old.");

}

}

// Using it:

Person p = new Person();

p.Name = "Vishnu";

p.Age = 24;

p.Speak();

## ✅ 8. Inheritance

csharp

CopyEdit

class Animal

{

public void Speak() => Console.WriteLine("Animal sound");

}

class Dog : Animal

{

public void Bark() => Console.WriteLine("Woof!");

}

## ✅ 9. Interfaces

csharp

CopyEdit

interface IDrive

{

void Drive();

}

class Car : IDrive

{

public void Drive()

{

Console.WriteLine("Driving a car");

}

}

## ✅ 10. Exception Handling

csharp

CopyEdit

try

{

int x = 5 / 0;

}

catch (DivideByZeroException e)

{

Console.WriteLine("Cannot divide by zero!");

}

finally

{

Console.WriteLine("This always runs");

}

## ✅ 11. Namespaces

Helps organize code.

csharp

CopyEdit

namespace MyApp

{

class Hello { }

}

## ✅ 12. Arrays and Lists

### Arrays

csharp

CopyEdit

int[] numbers = { 1, 2, 3 };

Console.WriteLine(numbers[0]); // 1

### Lists (using System.Collections.Generic)

csharp

CopyEdit

List<string> fruits = new List<string>();

fruits.Add("Apple");

Console.WriteLine(fruits[0]);

## 🧠 Summary Table

| Concept | Keyword / Syntax |
| --- | --- |
| Print output | Console.WriteLine() |
| Variable | int, string, bool, etc. |
| Function | static void MyFunc() {} |
| Condition | if, else |
| Loop | for, while |
| Class | class, object creation |
| Inheritance | class Child : Parent |
| Interface | interface, implements |
| Exception | try, catch, finally |
| Collection | Array, List<T> |

Exactly! ✅ **C# primitive types behave like objects — very similar to how they do in Python.**

Let’s compare it clearly between **C#**, **Python**, **Java**, and **JavaScript**:

## 🔄 1. Comparison of Primitive Type Behavior

| Language | Are primitives objects? | Notes |
| --- | --- | --- |
| **Python** | ✅ Yes | Everything is an object (int, str, bool, etc.) |
| **C#** | ✅ Yes (via boxing / structs) | Value types like int, double are actually structs with methods |
| **Java** | ❌ No | Primitives (int, char, etc.) need wrapper classes (Integer, Character) |
| **JavaScript** | ✅ Mostly | Primitives are auto-wrapped into object types when needed |

## ✅ 2. C# Primitives = Structs with Methods

In C#, types like int, bool, double, etc. are **structs**, not just "raw memory" values.

Example:

csharp

CopyEdit

int x = 10;

Console.WriteLine(x.ToString()); // "10"

Console.WriteLine(x.GetType()); // System.Int32

Here, int is an alias for System.Int32, which is a **struct** in .NET — meaning it **has methods** and behaves like an object in many ways.

## 🧠 3. What is Boxing?

Boxing is the process of **converting a value type (like** int**) into an object**, allowing it to be stored or used as an object.

csharp

CopyEdit

int x = 42;

object obj = x; // Boxing

int y = (int)obj; // Unboxing

So yes — primitives **behave like objects**, but they're still technically **value types**, not reference types (like classes).

## ✅ 4. Real-World Example

csharp

CopyEdit

int num = 123;

// All these are valid:

Console.WriteLine(num.ToString());

Console.WriteLine(num.CompareTo(100)); // Returns 1

Console.WriteLine(num.Equals(123)); // Returns true

Console.WriteLine(num.GetHashCode());

You don't need a special wrapper like Java's Integer because int is already a struct with methods.

## 🧪 TL;DR Summary

* ✅ **C# primitive types are object-like** (like Python), thanks to being structs.
* ✅ You can call methods on them directly.
* ⚠️ They are **value types**, not reference types — meaning they get copied when passed to methods, unlike class objects.

Let me know if you want a visual diagram of **value vs reference types**, or a practical code demo like how values change in memory!

# 🎯 ADVANCED C# CONCEPTS (Senior Dev Level)

## 1. ✅ Value Types vs Reference Types (Deep Dive)

* **Value types**: stored in **stack**, copied by value (e.g., int, struct)
* **Reference types**: stored in **heap**, copied by reference (e.g., class)
* Concepts: **boxing**, **unboxing**, **immutability**, **performance implications**

🧠 **Q: What’s the performance cost of boxing/unboxing and how to avoid it?**

## 2. ✅ Delegates, Func, Action, and Lambda Expressions

* Delegates: Type-safe method pointers.
* Func<>: Delegate that returns a value.
* Action<>: Delegate that returns void.
* Lambdas: Anonymous functions.

csharp

CopyEdit

Func<int, int, int> add = (a, b) => a + b;

🧠 **Q: When would you use a delegate over an interface?**

## 3. ✅ Events and Event Handling

* Custom event declaration
* Using .Invoke() safely
* EventHandler vs EventHandler<T>

csharp

CopyEdit

public event EventHandler<MyEventArgs> OnDataReceived;

## 4. ✅ LINQ Internals

* LINQ uses **deferred execution**.
* You can chain filters, projections, groupings.
* IEnumerable vs IQueryable

🧠 **Q: How does deferred execution work? When do queries actually run?**

## 5. ✅ Asynchronous Programming

* async/await, Task, Task<T>
* Deadlocks and ConfigureAwait(false)
* Parallelism vs Concurrency

🧠 **Q: What’s the difference between** Task.Run() **and** await**?**  
🧠 **Q: How would you avoid a deadlock in an ASP.NET app?**

## 6. ✅ Memory Management & Garbage Collection

* .NET uses **Generational GC** (Gen 0, 1, 2).
* Large Object Heap (LOH)
* Finalizers vs IDisposable

🧠 **Q: When to implement** IDisposable**?**  
🧠 **Q: What does** GC.Collect() **do and when should you avoid it?**

## 7. ✅ Reflection and Dynamic Code

csharp

CopyEdit

var type = typeof(MyClass);

var method = type.GetMethod("DoWork");

method.Invoke(instance, null);

* Use in plugin systems, dynamic loading.
* Performance impact of reflection.

## 8. ✅ Attributes and Annotations

* Custom attributes
* Reading attributes via reflection

## 9. ✅ SOLID Principles and Design Patterns

* Single Responsibility, Open-Closed, etc.
* Common patterns:
  + Singleton
  + Factory
  + Dependency Injection
  + Repository
  + Mediator

🧠 **Q: Why would you use Mediator over direct service calls in a controller?**

# 🔧 ADVANCED ASP.NET CORE CONCEPTS

## 1. ✅ Middleware Pipeline

* Request → Middleware1 → Middleware2 → Endpoint → Response
* app.Use(), app.Run(), app.Map()

🧠 **Q: What’s the difference between** Use**,** Run**, and** Map**?**

## 2. ✅ Dependency Injection (DI)

* Built-in DI container.
* Registering Transient, Scoped, Singleton services.

🧠 **Q: When would you use a scoped service over singleton?**  
🧠 **Q: How do you inject services into middleware or filters?**

## 3. ✅ Filters in ASP.NET

* Authorization Filters
* Resource Filters
* Action Filters
* Exception Filters

🧠 **Q: When to use middleware vs action filters?**

## 4. ✅ Model Binding & Validation

* Custom model binders
* Fluent Validation
* Data Annotations

## 5. ✅ Routing and Endpoints

* Conventional Routing
* Attribute Routing
* MapControllers, MapGet, etc.

🧠 **Q: How do you implement versioning in an ASP.NET API?**

## 6. ✅ Authentication & Authorization

* JWT Bearer tokens
* Claims-based auth
* Policies and Roles

🧠 **Q: How do you build a custom authorization policy?**

## 7. ✅ SignalR for Real-Time Communication

* WebSockets abstraction
* Hubs and Clients

🧠 **Q: How does SignalR manage connection IDs?**

## 8. ✅ Entity Framework Core (EF Core)

* DbContext lifecycle
* Lazy vs Eager loading
* Change tracking and shadow properties
* Raw SQL and stored procedures

🧠 **Q: How do you optimize EF Core for large datasets?**  
🧠 **Q: How would you implement Unit of Work and Repository with EF Core?**

## 9. ✅ Configuration and Environment Settings

* appsettings.json, IConfiguration
* Environment-based settings (Development, Production)

## 10. ✅ Testing ASP.NET Apps

* xUnit, Moq
* Testing controllers, services, middleware
* Integration vs unit testing

🧠 **Q: How do you mock HttpContext or IOptions?**

# 🧠 Pro Interview-Level Questions

1. **Explain the ASP.NET Core request lifecycle.**
2. **How does the garbage collector work in .NET and how do you profile memory?**
3. **What is the difference between** IEnumerable **and** IQueryable**?**
4. **What’s the impact of using** .Result **or** .Wait() **on async code?**
5. **How do you handle circular dependencies in DI?**
6. **How would you secure sensitive config data in a cloud-deployed ASP.NET Core app?**
7. **What are the best practices for exception handling in a large-scale .NET web app?**
8. **How would you implement distributed caching (like Redis) in ASP.NET Core?**
9. **How would you scale an ASP.NET Core app horizontally?**
10. **What performance tuning techniques do you use for high-traffic APIs?**

Let me know if you want:

* An **ASP.NET Core system design architecture**
* A **cheat sheet for async, DI, or filters**
* Or to **build and deploy a production-ready ASP.NET Core REST API** step-by-step.

You're close to senior-level prep. Let's get your edge sharpened.