### **1. Built-in IoC Container in ASP.NET Core Web API**

* **Theory**: The built-in IoC container manages the creation and resolution of dependencies, allowing you to inject services into controllers, services, etc.
* **Code**:
  + Register services in Program.cs using methods like AddSingleton, AddScoped, or AddTransient.
  + Resolve dependencies by injecting them into controllers or services.

builder.Services.AddSingleton<IMyService, MyService>();

### **2. Registering Application Services**

* **Theory**: You register application-specific services in the IoC container. This allows services like business logic, data access, etc., to be injected into controllers or other services.
* **Code**:
  + Register services in Program.cs.

builder.Services.AddScoped<IProductService, ProductService>();

### **3. Understanding Service Lifetime**

* **Theory**: Service lifetime defines how long an instance of a service is retained.
  + **Singleton**: One instance for the entire application.
  + **Scoped**: One instance per HTTP request.
  + **Transient**: A new instance for every request.
* **Code**:
  + Use AddSingleton, AddScoped, or AddTransient to define service lifetimes.

builder.Services.AddSingleton<IProductService, ProductService>(); // Singleton  
builder.Services.AddScoped<IProductService, ProductService>(); // Scoped  
builder.Services.AddTransient<IProductService, ProductService>(); // Transient

### **4. Extension Methods for Registration**

* **Theory**: Use extension methods to organize service registration logic, improving modularity and maintainability. Group related services together for cleaner Program.cs files.
* **Code**:
  + Define extension methods for service registration in a separate class.

public static void AddProductServices(this IServiceCollection services)  
{  
 services.AddTransient<IProductService, ProductService>();  
}

* Call the extension method in Program.cs.

builder.Services.AddProductServices();

### **5. Constructor Injection**

* **Theory**: Constructor Injection allows you to inject dependencies into classes via their constructors. This is the most common form of DI in ASP.NET Core.
* **Code**:
  + Inject dependencies in the constructor and let the IoC container handle the instantiation.

public class ProductController : ControllerBase  
{  
 private readonly IProductService \_productService;  
  
 public ProductController(IProductService productService)  
 {  
 \_productService = productService;  
 }  
}

### **Summary of Key Concepts**

1. **IoC Container**: Centralized container that resolves and injects dependencies.
2. **Service Registration**: Add services to the IoC container with the desired lifetime (Singleton, Scoped, Transient).
3. **Service Lifetime**: Defines the lifespan and scope of services in the container.
4. **Extension Methods**: Used to modularize and organize service registration for cleaner code.
5. **Constructor Injection**: A way to inject dependencies into classes via their constructor for better decoupling and easier testing.