**C# Notes**

*Intro Basis (Q to ChatGPT):*  
With regards to C# Automation API and Back End Testing, if you look at all the relevant topics of C# software development programming, understanding and implementation and all the categories and topics that a person needs to have an understanding of. This will all range from Beginner, to Intermediate to Advanced levels.

Give me a list of all the skills, topics, categories, etc that a person needs and split this into the 3 levels (Beginner, Intermediate and Advanced).

Your list needs to be all encompassing to handle all areas needed to be contributing to a team's code base asap.

*Response from ChatGPT:*

Here’s a **comprehensive roadmap** split by **Beginner**, **Intermediate**, and **Advanced** levels, specifically tailored for a developer aiming to contribute to a **C# Automation API and Backend Testing** project.

This guide covers the **C# language**, **automation**, **API testing**, **back-end fundamentals**, and the **supporting ecosystem/tools** required to be productive in a modern .NET team environment.

Here’s a **comprehensive roadmap** split by **Beginner**, **Intermediate**, and **Advanced** levels, specifically tailored for a developer aiming to contribute to a **C# Automation API and Backend Testing** project.

This guide covers the **C# language**, **automation**, **API testing**, **back-end fundamentals**, and the **supporting ecosystem/tools** required to be productive in a modern .NET team environment.

**✅ Beginner Level**

**Goal**: Understand the fundamentals of C#, .NET, basic testing concepts, and essential tooling.

**🧠 C# Language Basics**

* Variables, data types, and operators
* Control structures (if, switch, loops)
* Functions and parameters
* Error handling (try/catch/finally)
* Classes and objects
* Namespaces and assemblies
* Basic debugging

**🧰 .NET & Project Structure**

* Introduction to .NET Core / .NET 6+
* Project types: Console app, Class Library, ASP.NET Web API
* Using Visual Studio / VS Code
* Project structure & solution organization

**🧪 Unit Testing Basics**

* Introduction to unit testing
* xUnit / NUnit basics
* Writing basic test methods
* Test project structure
* Running tests from IDE/CLI

**🌐 Web & APIs**

* Understanding HTTP methods (GET, POST, PUT, DELETE)
* REST vs SOAP
* Consuming APIs using HttpClient
* JSON serialization/deserialization with System.Text.Json or Newtonsoft.Json

**📦 NuGet and Package Management**

* Installing and managing NuGet packages
* Common packages: xUnit, RestSharp, FluentAssertions

**💻 Basic Version Control (Git)**

* Cloning a repo
* Basic commands: add, commit, push, pull
* Branching and merging

**🚀 Intermediate Level**

**Goal**: Build solid backend test automation using real-world patterns, contribute to a codebase, and understand best practices.

**💻 Intermediate C# Concepts**

* OOP principles (Encapsulation, Inheritance, Polymorphism)
* Interfaces and abstraction
* Generics
* Enums, Structs
* LINQ
* Collections and dictionaries
* File I/O basics

**🔍 Test Automation Skills**

* Writing modular and reusable tests
* Test setup and teardown methods
* Data-driven testing
* Mocking (using Moq or NSubstitute)
* FluentAssertions or similar assertion libraries
* Using test runners in CI pipelines

**🌐 API Testing**

* Automated testing of RESTful APIs
* Using RestSharp / HttpClient with test frameworks
* Authentication & Authorization (Bearer tokens, Basic Auth)
* Postman collections to C# tests migration
* Contract validation (JSON schema)

**🛠 Test Frameworks & Tools**

* xUnit/NUnit: Fixtures, Theory/Data tests
* RestSharp for API interactions
* WireMock.Net or MockHttp for API mocking
* Serilog or similar logging for test output

**🏗 Backend System Concepts**

* RESTful API design principles
* HTTP Status codes & headers
* JSON / XML formats
* Basic understanding of databases (SQL Server, SQLite)
* Using EF Core or Dapper (read-only/test context)

**🧪 Integration Testing**

* API-level integration tests
* Setting up test environments
* Working with in-memory databases (InMemoryProvider, SQLite)
* Test doubles (Fakes, Stubs, Mocks)

**⚙️ Dependency Injection (DI)**

* Understanding DI in .NET Core
* Configuring services for testing
* Overriding services in tests

**📂 Source Control & Collaboration**

* Understanding PR workflows
* Code review basics
* Resolving merge conflicts
* Using GitHub Actions or Azure DevOps for CI

**🧠 Advanced Level**

**Goal**: Take full ownership of test architecture, mentor others, and improve team productivity through tooling and strategy.

**💡 Advanced C# Topics**

* Asynchronous programming (async/await, Task)
* Delegates, Events, Func, Action
* Expression Trees and Reflection
* Extension Methods
* Attributes and Annotations
* Memory management and performance tuning

**🧪 Advanced Testing Practices**

* Test architecture (layered testing: Unit, Integration, E2E)
* Custom assertions
* Building internal test frameworks/libraries
* Contract testing (e.g. using PactNet)
* Test parallelization
* Snapshot testing
* Load and performance testing with k6 or NBomber

**🔐 Security & Auth Testing**

* OAuth 2.0 / OpenID Connect flows
* Testing APIs with JWT tokens
* Role-based access testing

**🛠 Advanced Tools & Libraries**

* AutoFixture for test data generation
* FluentValidation for validation testing
* SpecFlow or BDD frameworks
* Dockerized testing environments
* Using Selenium or Playwright for API + UI flows

**🧬 CI/CD Integration**

* Writing test stages in CI/CD pipelines (YAML)
* Managing test reports (HTML, Allure, TRX)
* Test result publishing in Azure DevOps/GitHub Actions
* Environment configuration (test/staging/prod)

**🧱 Architecture & Design Patterns**

* Clean Architecture
* SOLID principles
* Repository pattern
* Builder, Factory, Strategy patterns
* Testability in code design

**📊 Observability & Metrics**

* Logging strategies in tests
* Monitoring test health/trends
* Integration with tools like Application Insights or Elastic Stack

**👥 Team Leadership & Contributions**

* Enforcing code standards in tests
* Driving test automation strategy
* Mentoring juniors
* Performing code reviews for test quality

**🧩 Optional / Cross-cutting Topics (All Levels)**

These can appear at any level depending on the project:

* YAML / JSON configurations
* Secrets management in test runs
* Using environment variables for test config
* REST API documentation tools (Swagger/OpenAPI)
* Message queues (RabbitMQ, Azure Service Bus – for back-end integration testing)
* Feature flags and testing around them

*Resources Encountered:*

Mocks:

<https://thecodebuzz.com/unit-test-mock-httpclientfactory-moq-net-core/>

<https://nsubstitute.github.io/>

<https://github.com/devlooped/moq/wiki/Quickstart>

GitHub CodeBase examples:  
<https://github.com/restsharp/RestSharp/>

Framework Documentation:  
<https://docs.nunit.org/articles/nunit/writing-tests/attributes/testcase.html>

YouTube C# Course (not sure of viability):  
<https://www.youtube.com/watch?v=GhQdlIFylQ8>

Assertion Library (Fluent):  
<https://fluentassertions.com/introduction>

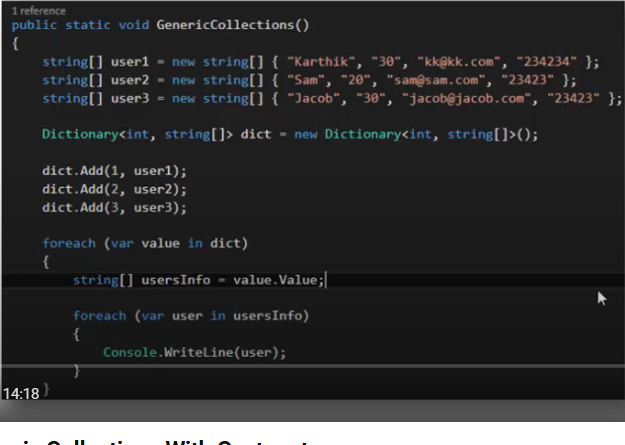
RestSharp Documentation:  
<https://restsharp.dev/docs/advanced/serialization>

Serialization and Deserialization Documentation:

<https://www.newtonsoft.com/json/help/html/SerializingJSON.htm>

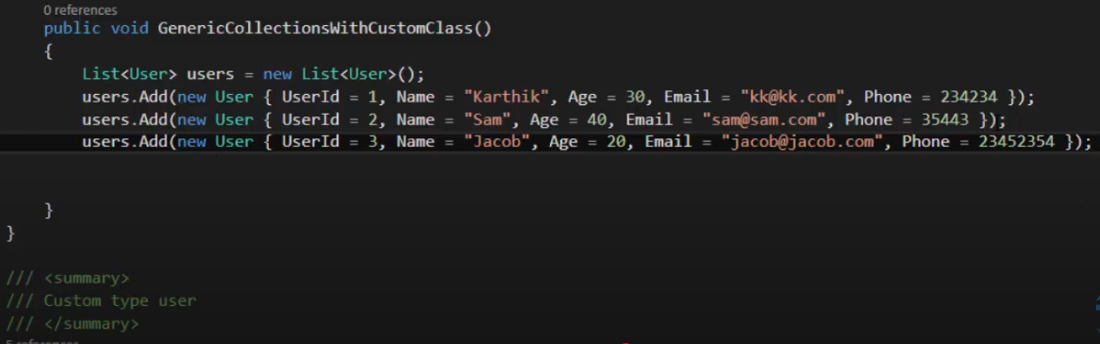
(need to find for System.Text.Json docs) <https://learn.microsoft.com/en-us/dotnet/api/system.text.json?view=net-9.0>

TODO:

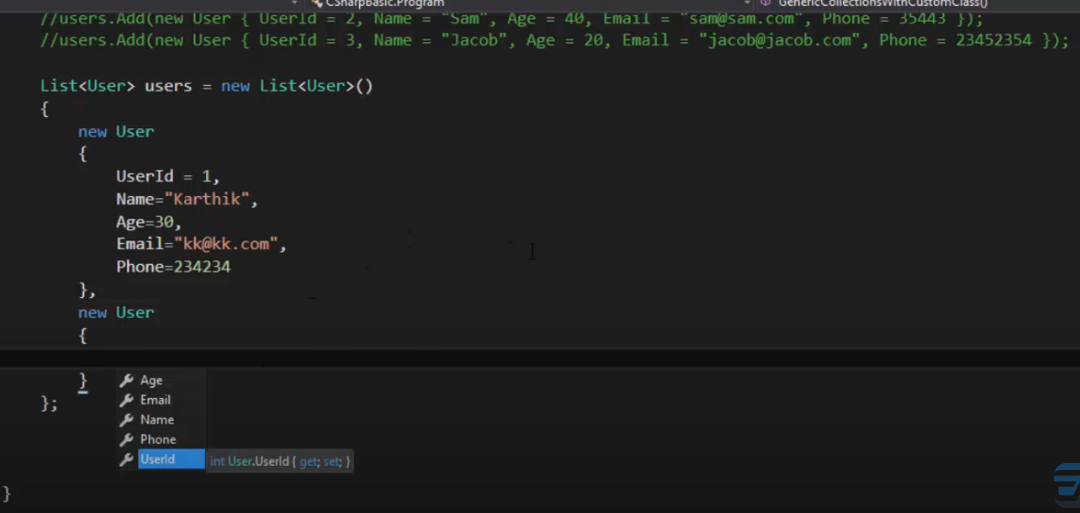
* + Non generic collection
  + Types of generic<> collection:
    - List<>
    - Dictionary<Tkey,Tvalue>
    - Stack<T>
    - E.g. 
  + How to use Generic Collections with Custom Types? Converting the above example to use a Custom Type = User.
    - E.g.

A screen shot of a computer program

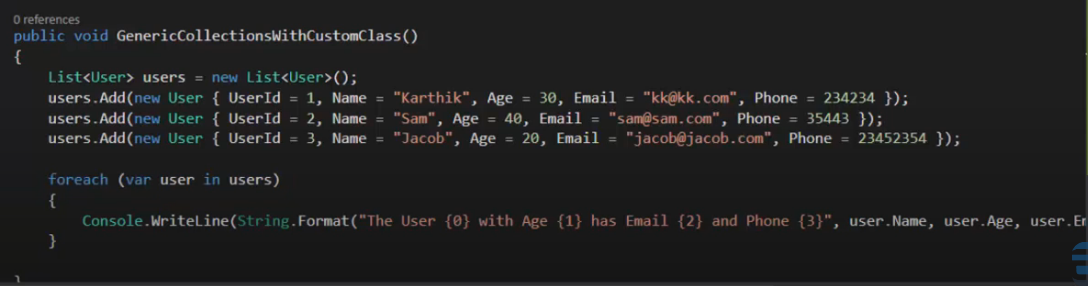
AI-generated content may be incorrect.



Or alternatively…



And, if you needed to iterate through the dictionary with the custom types…



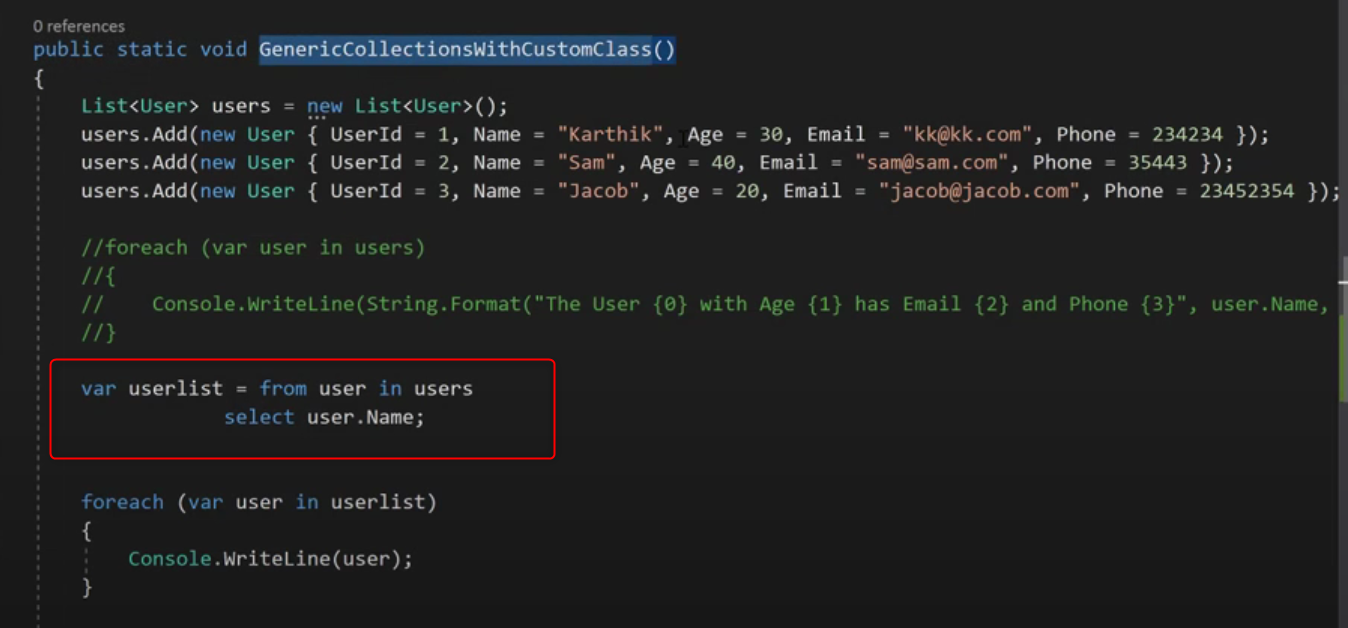
* + LINQ Query (this is the most common type of extension method that adds query functionality to the existing System.Collections.IEnumerable() and System.Collections.Generic.IEnumerable<T> types.

LINQ Query Notes:

2 ways of executing LINQ: Query Based and Method Based.

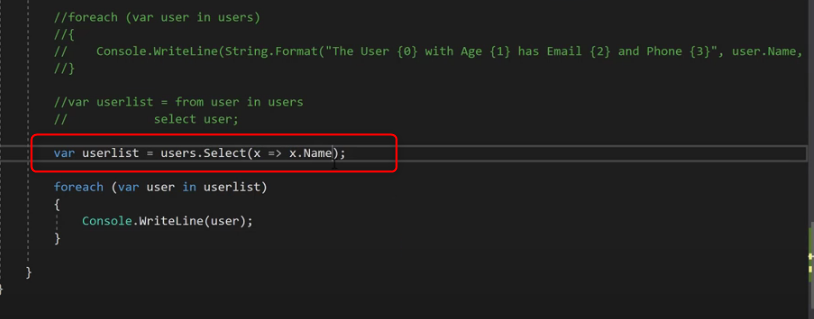
**Example of Query Based approach:**

From the example used earlier, the foreach loop is commented out and the query based code approach is within the red block. We then use a foreach loop, to iterate through the query based variable.

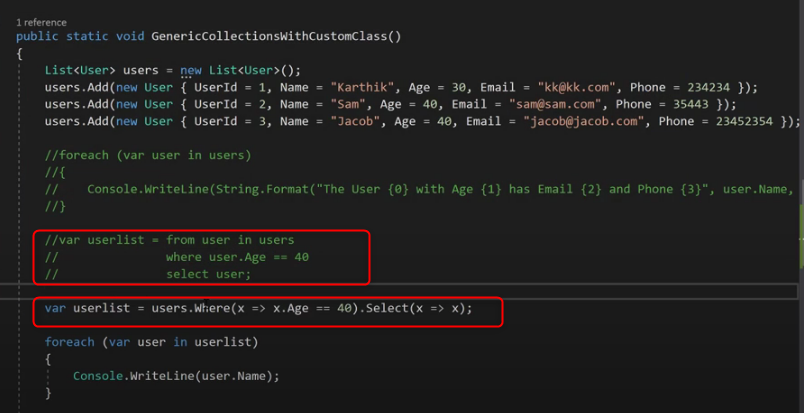


**Example of Method Based approach:**

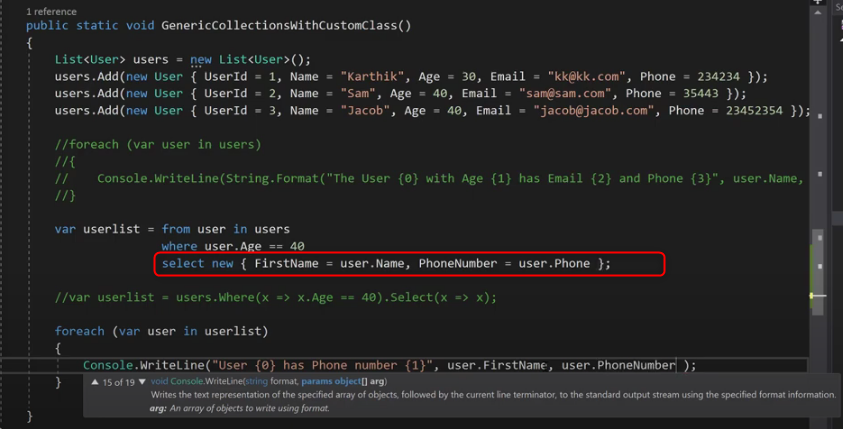
The query bas ed method is commented out and implementation is replaced with the method based approach within the red block.



If you need to tweak further to get more specific information using a **where clause**. Red blocks below indicate code syntax for both approaches.



Example below is how to use Anonymous Types in a LINQ:



End of LINQ Query points.

**🟢 Generic Collections (modern style)**

These collections are **type-safe** — you tell them up front **what type of data** they will hold.

**🔧 Example:**

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

names.Add("Alice");

names.Add("Bob");

* Now the list can **only hold strings**.
* You avoid errors and your code is easier to understand and test.

**✅ How to Use Generics in Test Scenarios**

Suppose you’re testing form inputs with NUnit:

[Test]

public void ValidateNames()

{

List<string> testNames = new List<string> { "John", "Jane", "Jill" };

foreach (string name in testNames)

{

Assert.IsNotEmpty(name);

}

}

* Easy to manage a list of test data.
* Easy to loop through and test each item.
* **Type-safe**: only works with strings.

**🔹 2. IEnumerable**

**🟦 What is IEnumerable?**

Think of IEnumerable like a **"read-only list"** or **"something you can loop over"**.

It’s **not a list itself**, but a **way to access one item at a time** in any collection that supports it.

**🧠 Example (Simple View):**

IEnumerable<string> fruits = new List<string> { "Apple", "Banana", "Cherry" };

foreach (var fruit in fruits)

{

Console.WriteLine(fruit);

}

You can **loop over** IEnumerable, but **you can’t add or remove** items.

**✅ Why use IEnumerable?**

* It's useful when you want to **just read** data, not modify it.
* It’s **lighter** than a full List<T>, good for performance.
* Many **LINQ queries** return IEnumerable<T> — more on this shortly.

**🧪 How it fits in NUnit:**

You can return test data using IEnumerable:

public static IEnumerable<string> ValidNames()

{

yield return "John";

yield return "Jane";

yield return "Joe";

}

[Test, TestCaseSource(nameof(ValidNames))]

public void Name\_ShouldNotBeEmpty(string name)

{

Assert.IsNotEmpty(name);

}

**🔹 3. LINQ (Language Integrated Query)**

**🟦 What is LINQ?**

LINQ lets you **filter, search, and manipulate data** using simple and readable code — similar to querying a database, but in memory.

Think of it as asking questions like:

“From this list, give me only names that start with J.”

**🔧 Simple Example:**

List<string> names = new List<string> { "Jack", "Bob", "Jill", "Sam" };

var jNames = names.Where(name => name.StartsWith("J"));

foreach (var name in jNames)

{

Console.WriteLine(name); // Jack, Jill

}

* Where() is a **LINQ method** that filters data.
* It returns an IEnumerable<string>

**✅ Why is LINQ useful?**

* **Cleaner code** for filtering or transforming data
* Works on **any collection** (lists, arrays, etc.)
* Frequently used in test scenarios to:
  + Filter test data
  + Validate specific conditions

**🧪 NUnit Example Using LINQ**

[Test]

public void Should\_OnlyContainNamesStartingWithJ()

{

List<string> names = new List<string> { "John", "James", "Bob" };

var result = names.Where(n => n.StartsWith("J")).ToList();

Assert.That(result, Is.All.StartsWith("J"));

}

**✅ Summary Table**

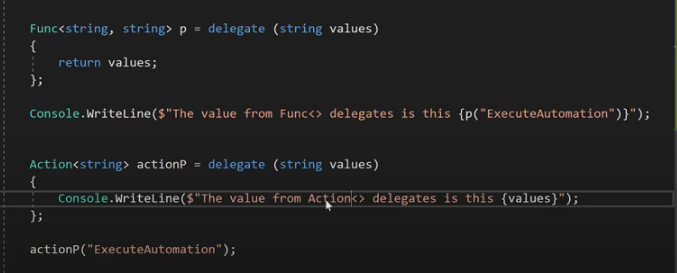
| **Concept** | **What it is** | **When to use it** | **Why it's useful** |
| --- | --- | --- | --- |
| **Generic List** | List that holds specific type (List<T>) | Anytime you store multiple known-type items | Type-safe, avoids errors |
| **Non-Generic** | Old-style list that holds anything | Avoid unless maintaining old code | Can cause runtime errors |
| **IEnumerable** | Interface for looping through items | Reading/iterating items without modification | Lightweight, works with LINQ |
| **LINQ** | Query-like syntax for filtering data | When filtering or transforming collections | Cleaner, more powerful data manipulation |

***NB: Lambda expressions is the current preferred way of interacting with Delegates at the moment.***

**NB: Need to explore Func or Function Delegates and Action Delegates of C# and then the current accepted practice of Lambda expressions interacting with Delegates.**

Below is an example of Function delegate syntax which will ALWAYS return something, Action delegate syntax that will NEVER return anything and Lambda expression syntax.

First screenshot is the syntax for Function Delegate and Action Delegate:



Below is the syntax for Lambda expression Delegate example, based on the conversion of the above data:

Action<string> actionWithLambda = values => {Console.WriteLine($”The value from Action<> lambda is: {values}”); };

actionWithLambda(“ExecuteAutomation”);

End of Delegate points.

**Next Topic: Tuples, OUT Keyword and Throwing Specific Exceptions**

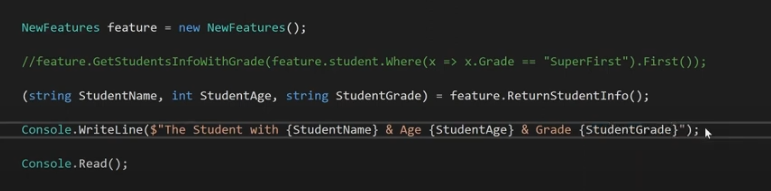
* + Allows you to return multiple return values from a method.
  + Is a more optimal approach then compared to using Out parameters and Anonymous types returned through a dynamic return.

NB: If Tuples is not added as part of the .NET Framework latest version, you will need to add it as a Nuget Package under References. The name is: System.ValueTuple. The Author is: Microsoft. **DON’T USE: System.Tuples**.

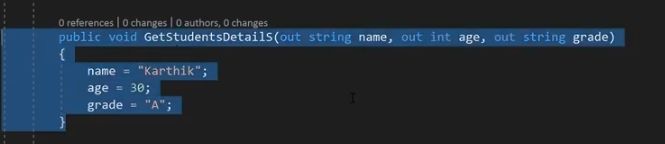
Code syntax to create:  
A screen shot of a computer

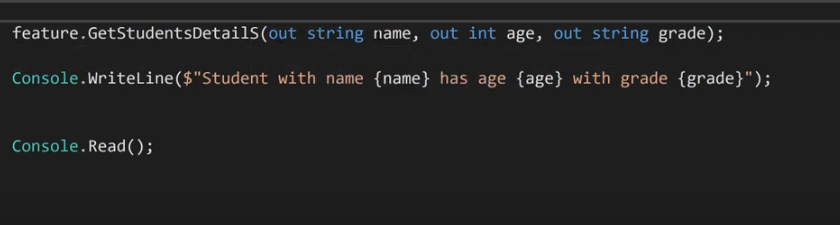
AI-generated content may be incorrect.

Code syntax to call: Because there are multiple returns for this one method, tuples allows you to assign those returns into variables.

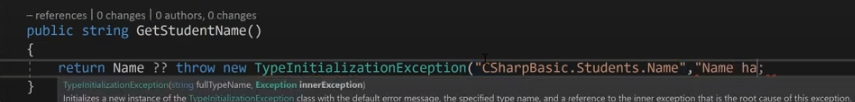


Another variation of tuple which might still appear in some legacy code is usage of OUT keyword, as shown below:





Throwing an exception when a property is null:



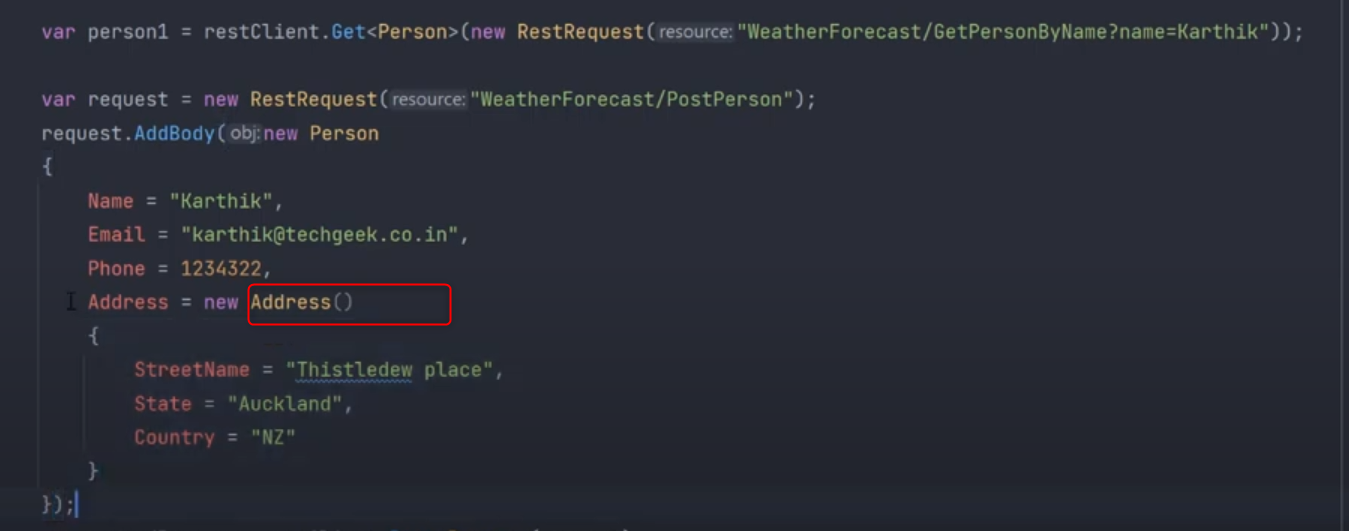
A computer screen with text

AI-generated content may be incorrect.

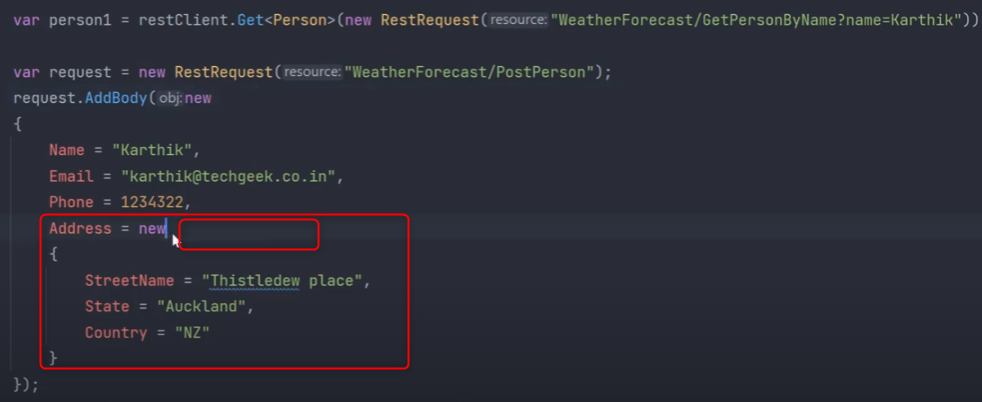
**NB: The concepts of records need to be explored later.**

Anonymous Types:

Code snippet below shows you usage of Address. This is an example of a concrete type.



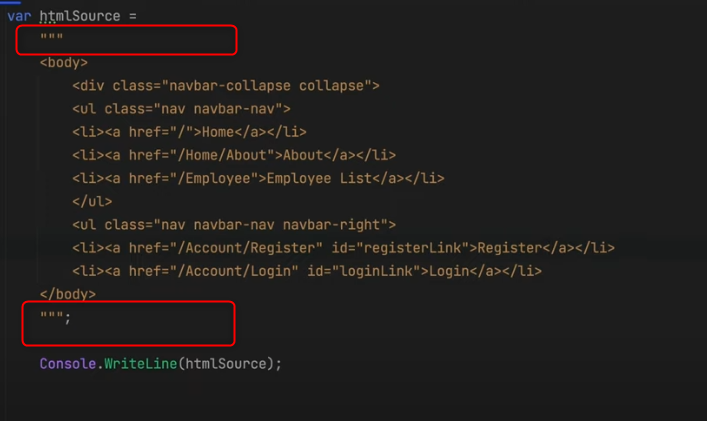
Anonymous Types is the removal of this, as shown below (the code will still work as normal):



Expansion of String Expression - Raw String Literal:

Scenario: Needed to use html body or even a json packet within a string. Wont be possible because of the syntax structure causing an issue with the usage of double quotes of the string.

Solution: You need to use double quotes (x3) “”” and these MUST be placed on a new line for it to work – as shown below: (This will work for html and json packets too).



Scenario: You wanted to use string interpolation with a json packet expression. See below.

