

Masterclass Repository

- ▶ <https://github.com/AThraen/HackYourFuture.NETMasterClass/>
- ▶ Curriculum
- ▶ Exercises
- ▶ Slides
- ▶ Sample solutions
- ▶ Videos
- ▶ Additional Reading

Session #1

IT ALL BEGINS HERE

Agenda for today

- ▶ Some basic concepts
- ▶ Introduction to the project we will code
- ▶ Food
- ▶ Coding the Console version – while introducing OOP
- ▶ Everybody codes

Master Class #1

After this session (and the accompanying homework) you should know:

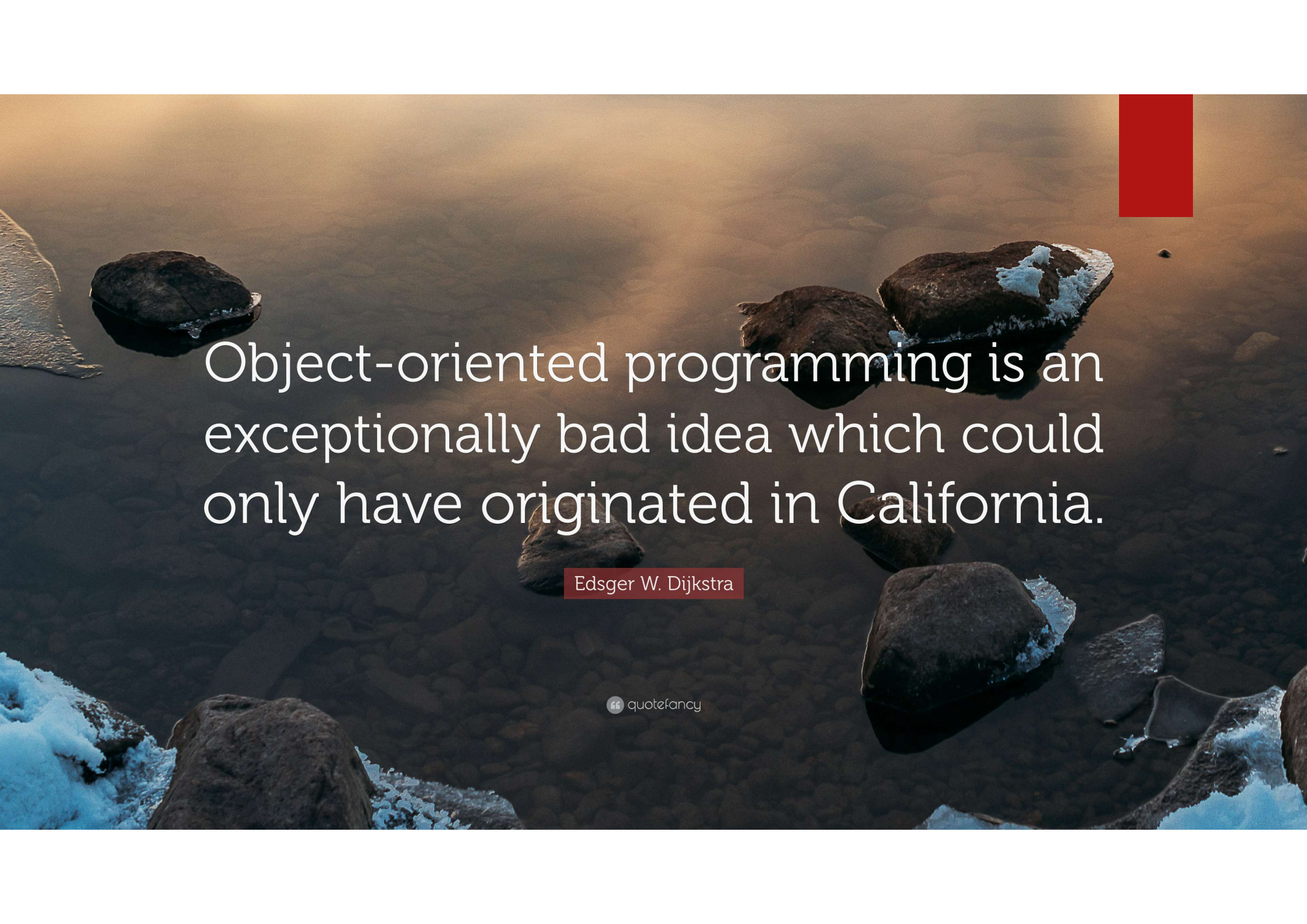
- ▶ *How to create a new solution/project in visual studio*
- ▶ *Understand the basics of Object Oriented Programming, including Inheritance, methods, properties, events and interfaces*
- ▶ *Understand what .NET Core is*
- ▶ *Be able to create a basic console application*
- ▶ *Use Visual Studio to debug your code*



Some basic concepts



Object Oriented Programming



Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger W. Dijkstra

OOP VS Functional Programming

Comparison Chart

OOP	Functional Programming
OOP is based on the concept of objects, instead of just functions and procedures.	It emphasizes on the use of function calls as the primary programming construct.
OOP follows the imperative programming model.	It's tightly connected to declarative programming.
It does not support parallel programming.	It supports parallel programming.
Basic elements are objects and methods.	Basic elements are variables and functions.
OOP brings together data and its associated behavior in a single location.	Data and its associated behavior are considered different entities and should be kept separate.

Object Oriented Programming (OOP)

```
public abstract class Vehicle
{
    protected bool _isDriving;

    public abstract int MaxPassengers {
        get;
    }
}

public class Bus : Vehicle
{
    public override int MaxPassengers {
        get { return 40; }
    }
}
```

```
public class Car : Vehicle
{
    //Members
    private int _fuelRemaining;


    //Properties
    public override int MaxPassengers { get { return 5; } }
    public string Make { get; private set; }
    public string Model { get; private set; }

    //Events
    public event Action<Car> HasStarted;
    public event Action<Car> HasStopped;

    //Methods
    public void Drive()
    {
        _isDriving = true;
        _fuelRemaining = _fuelRemaining - 1; // _fuelRemaining--;
        HasStarted?.Invoke(this);
    }

    public void Stop()
    {
        _isDriving = false;
        HasStopped?.Invoke(this);
    }

    //Constructor
    public Car(string Make, string Model)
    {
        this.Make = Make; this.Model = Model;
        _isDriving = false; _fuelRemaining = 100;
    }
}
```



```
Car car = new Car("Fiat", "500");  
car.HasStarted += c => Console.WriteLine(c.Make + " " + c.Model + " has started");  
car.Drive();
```

Encapsulation

Abstraction

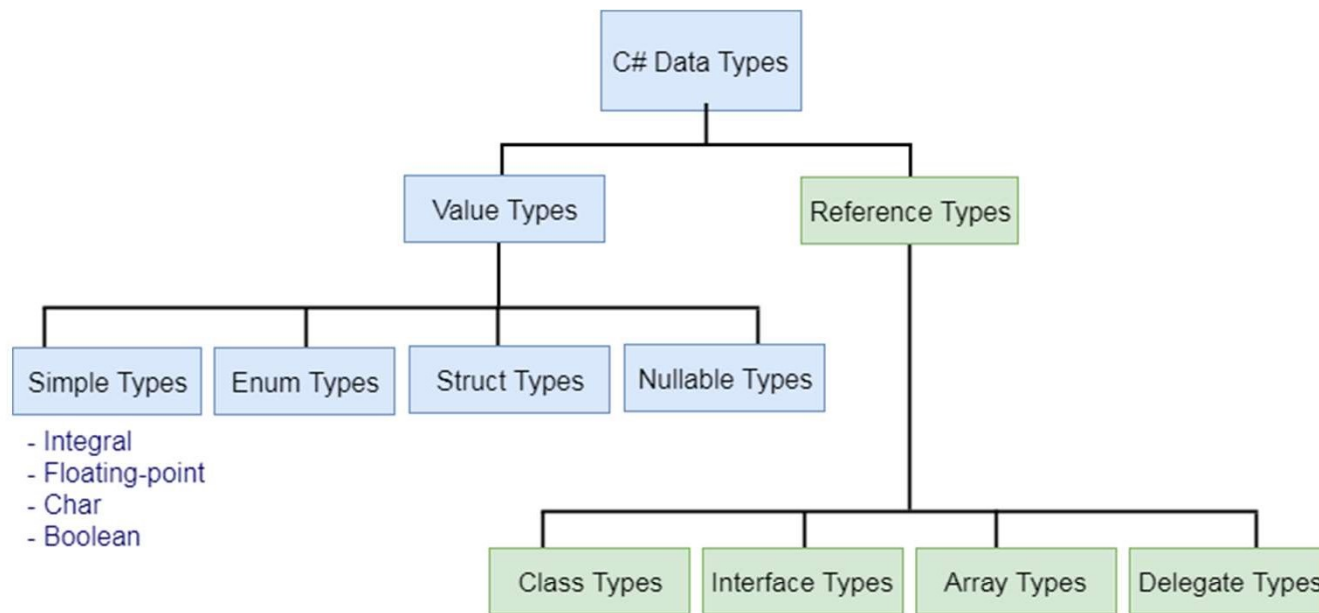
OOP

Inheritance

Polymorphism

<https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727>

Strong Typing



Types

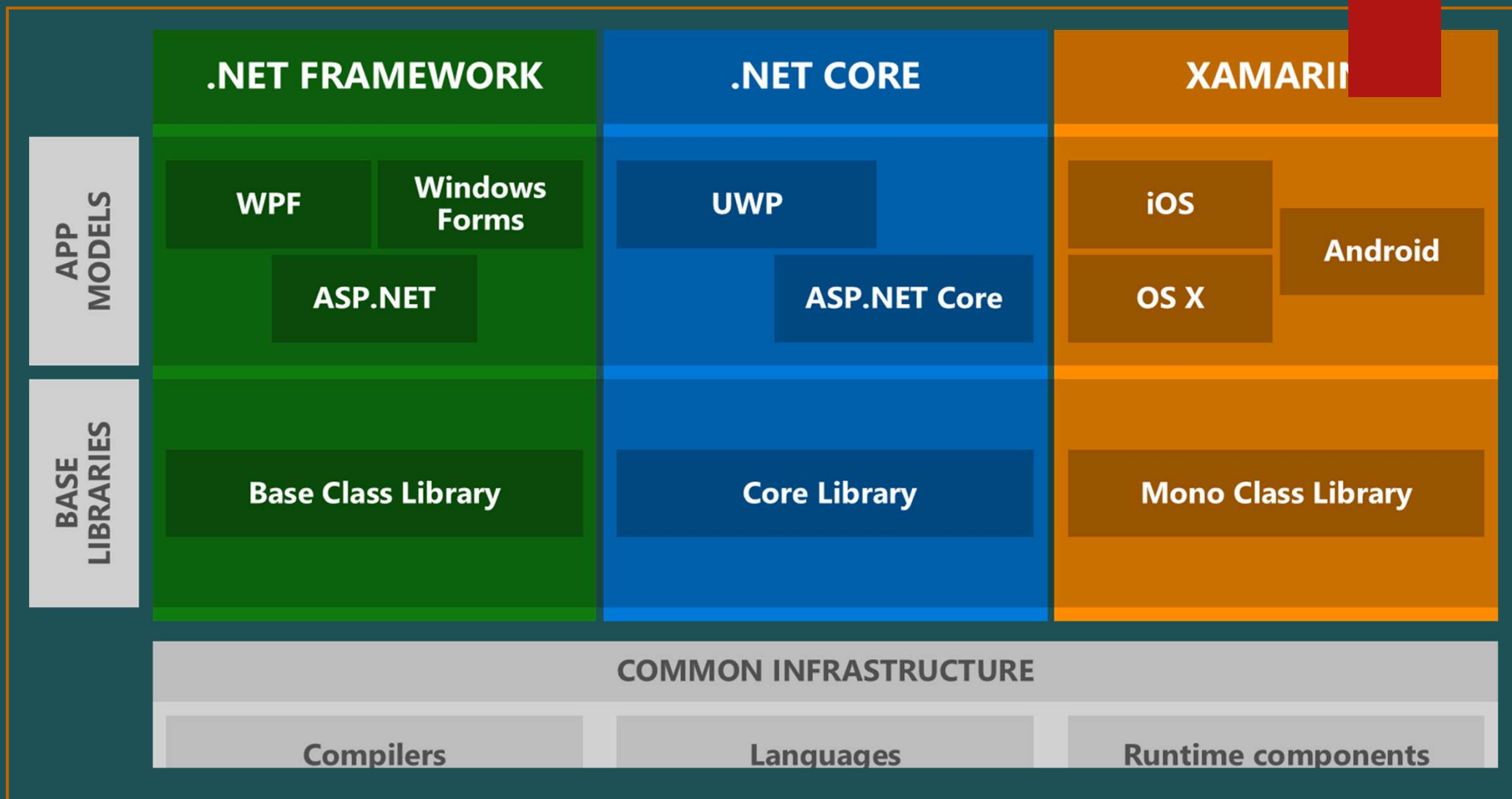
<https://www.tutorialsteacher.com/csharp/csharp-data-types>

A SIDE-BY-SIDE COMPARISON OF COMPILED LANGUAGES AND INTERPRETED LANGUAGES

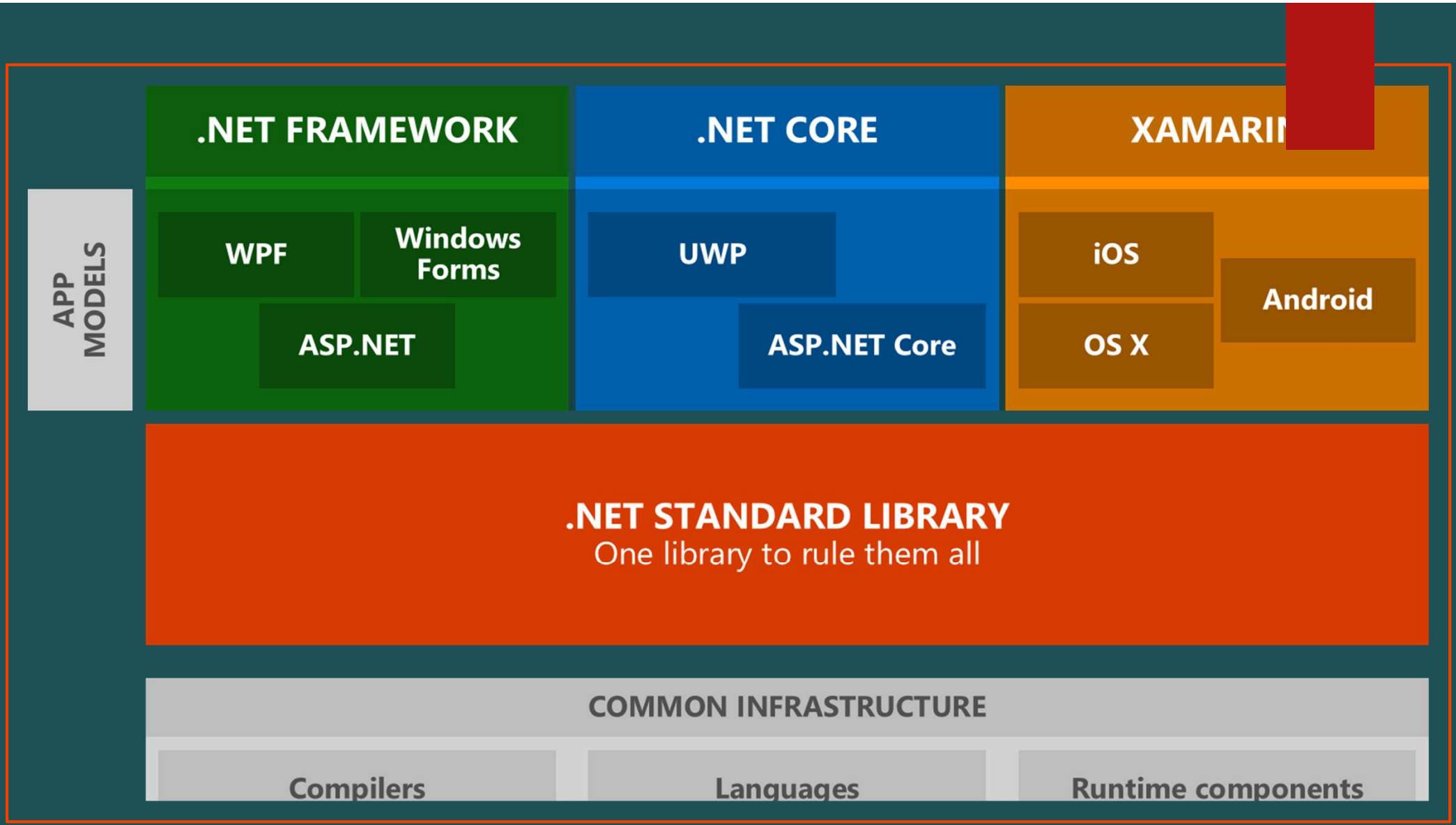
A look at how compilers and interpreters work, and how their differences affect memory, runtime speed, and computer workload.

	A COMPILER	AN INTERPRETER
Input	... takes an entire program as its input.	... takes a single line of code, or instruction, as its input.
Output	... generates intermediate object code.	... does not generate any intermediate object code.
Speed	... executes faster.	... executes slower.
Memory	... requires more memory in order to create object code.	... requires less memory (doesn't create object code).
Workload	... doesn't need to compile every single time, just once.	... has to convert high-level languages to low-level programs at execution.
Errors	... displays errors once the entire program is checked.	... displays errors when each instruction is run.

Compiled vs Interpreted languages



<https://devblogs.microsoft.com/dotnet/introducing-net-standard/>



IDE vs Editor

Integrated Development
Environment



Advanced & powerful Editor



Visual Studio Code

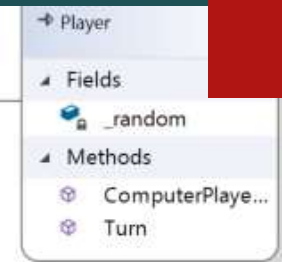
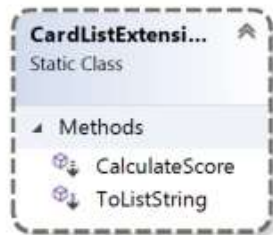
Project Types / Templated projects

- ▶ Console
- ▶ Web - MVC
- ▶ Windows Forms
- ▶ Universal Windows
- ▶ Extensions



Example game: Simplified 31

- ▶ Each player get's 3 cards
- ▶ Ace counts as 11, Jacks, Queens and Kings as 10, the rest as their value.
- ▶ First player to reach 31 in same suit wins
- ▶ Players take turns picking up / dropping cards from the Deck or the table
- ▶ A player can knock (call) in stead of taking a turn if they think they have a higher score than the others



Final Goal

QUICK DEMO

Code time

Homework after session #1

- ▶ Complete working console game
- ▶ Exercise descriptions and sample solution available in repository