Become a **Master** in **ASP.NET** (from **Scratch**)

0

Importance of Programming

https://www.youtube.com/watch?v=Dv7gLpW91DM

Hello!

I am Jahanzeb Naeem

I am here because I have 14+ years experience in software development mainly .NET.

Running my software house for last 4+ years.

MPhil from UMT.

Interest in Development (any language), AI, Machine Learning and Bigdata Analysis.

1

Introductions

Kindly introduce yourself.

You can introduce yourself in Urdu / English.

Required for Introduction:

- Full Name
- Background (Matric/FSC)
- Major (CS, SE, IT, etc.)

Success is the result of perfection, hard work, *learning* from failure, loyalty, and persistence.

“

2

An Introduction to C#

Let's start with the second set of slides

What is Visual Studio?

- Is an Integrated Development Environment by Microsoft.
- What is an IDE?

What is .NET?

- Is free
- Cross-platform
- Open source developer platform for building many different types of applications.
- Languages
  - C#, F# or Visual Basic.

What is .NET (Cont.)

- Consists of Mainly Two things:
  - FCL (Framework Class Libraries)/Readymade Classes.
    - Classes which are written by Microsoft as a prewritten code for standard works.
  - CLR (Common Language Runtime)
    - Is for:
      - Garbage Collection
      - Code Access Security
      - Code Verification
      - Intermediate Code (using JIT)

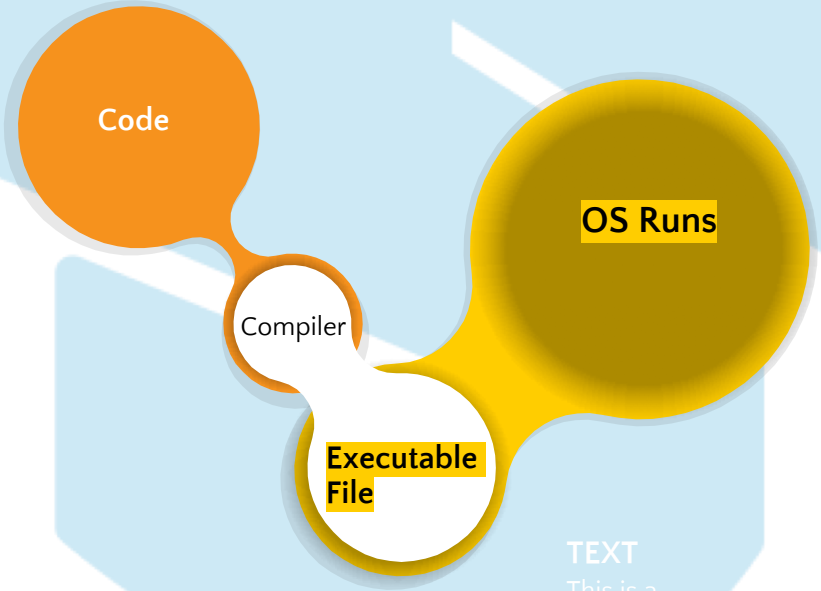
Why .NET?

- Have Readymade Classes.
- Reusability
- Using OOP, make huge projects
- Compatibility with new hardware is very easy.
- Make different types of apps.

Other important .NET Concepts

- MSIL (Microsoft Intermediate Language)
- CTS (Common Type System)
  - CLS (Common Language Specification)
- Assembly
  - Unit of Deployment.
  - Is self describing.
  - Manifest, contains all metadata about it.

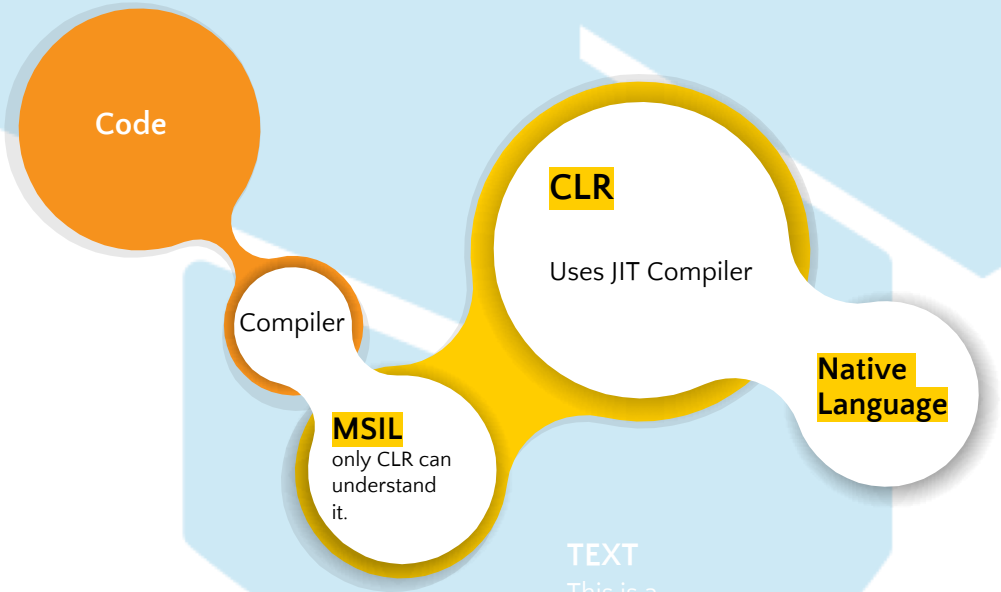
Code execution cycle – Old



SAMPLE TEXT

TEXT
This is a sample text

Code execution cycle – .NET



SAMPLE TEXT
This is a sample

TEXT
This is a sample text

What Data Types are available in .NET?

- Two types of Data:
  - Value Types
  - Reference Types
  - Value Types
    - Smaller in Size
    - Copy of original
    - Stack Declaration
    - Not handled by GC (Garbage Collector)
    - Example:
      - Primitive Data Types, Enumerations, Struct.

What Data Types are available in .NET?

- Reference Types
  - Larger Size
  - Nickname/Reference
  - Heap Declaration
  - Handled by GC (Garbage Collector)
  - Examples:
    - Classes, Interfaces, Delegates etc.

What are Primitive Types?

- Simple Types
- Boolean (bool)
  - Boolean data type has two possible values: true or false
  - Default value: false
  - Boolean variables are generally used to check conditions such as in *if statements*, *loops*, etc.

What are Primitive Types – (Cont.)

- Signed Integral
  - These data types hold integer values (both positive and negative).
  - Out of the total available bits, one bit is used for sign.
  - **sbyte**
    - Size: 8 bits
    - Range: -128 to 127.
    - Default value: 0

What are Primitive Types – (Cont.)

- **short**
  - Size: 16 bits
  - Range: -32,768 to 32,767
  - Default value: 0
- **int**
  - Size: 32 bits
  - Range: -231 to 231-1
  - Default value: 0

What are Primitive Types – (Cont.)

- long
  - Size: 64 bits
  - Range: -263 to 263-1
  - Default value: 0L [L at the end represent the value is of long type]

What are Primitive Types – (Cont.)

- Unsigned Integral
  - These data types only hold values equal to or greater than 0.
  - We generally use these data types to store values when we are sure, we won't have negative values.
  - **byte**
    - Size: 8 bits
    - Range: 0 to 255.
    - Default value: 0

What are Primitive Types – (Cont.)

- **ushort**
  - Size: 16 bits
  - Range: 0 to 65,535
  - Default value: 0
- **uint**
  - Size: 32 bits
  - Range: 0 to 232–1
  - Default value: 0

What are Primitive Types – (Cont.)

- **ulong**
  - Size: 64 bits
  - Range: 0 to 264–1
  - Default value: 0

What are Primitive Types – (Cont.)

- Floating Point
  - These data types hold floating point values i.e. numbers containing decimal values.
  - For example, 12.36, -92.17, etc.
  - **float**
    - Single-precision floating point type
    - Size: 32 bits
    - Range: 1.5 × 10−45 to 3.4 × 1038
    - Default value: 0.0F [F at the end represent the value is of float type]

What are Primitive Types – (Cont.)

- **double**
  - Double-precision floating point type.
  - Size: 64 bits
  - Range: 5.0 × 10−324 to 1.7 × 10308
  - Default value: 0.0D [D at the end represent the value is of double type]

What are Primitive Types – (Cont.)

- **Character (char)**
  - It represents a 16 bit unicode character.
  - Size: 16 bits
  - Default value: '\0'
  - Range: U+0000 ('\u0000') to U+FFFF ('\uffff')

What are Primitive Types – (Cont.)

- **Decimal**
  - Decimal type has more precision and a smaller range as compared to floating point types (double and float).
  - It is appropriate for monetary calculations.
  - Size: 128 bits
  - Default value: 0.0M [M at the end represent the value is of decimal type]
  - Range: (-7.9 x 1028 to 7.9 x 1028) / (100 to 28)

What are Namespace

- .NET Framework uses namespaces to organize its many classes
- Namespaces have the following properties:
  - They organize large code projects.
  - They are delimited by using the **.** operator.
  - The **using** directive obviates the requirement to specify the name of the namespace for every class.
  - The **global** namespace is the "root" namespace: **global::System** will always refer to the .NET **System** namespace.

What are Statements, Expressions and Operators?

- Statements
  - The actions that a program takes are expressed in statements.
  - Common actions include declaring variables, assigning values, calling methods, looping through collections, and branching to one or another block of code, depending on a given condition.
  - A statement can consist of a single line of code that ends in a semicolon, or a series of single-line statements in a block.

What are Statements, Expressions and Operators – (Cont.)

- Expressions
  - Is a sequence of one or more operands and zero or more operators that can be evaluated to a single value, object, method, or namespace.
  - Can consist of a literal value, a method invocation, an operator and its operands, or a **simple name**. Simple names can be the name of a variable, type member, method parameter, namespace or type.

What are Statements, Expressions and Operators – (Cont.)

- Operators
  - An **operator** is a program element that is applied to one or more **operands** in an expression or statement.
  - Operators that take one operand, such as the increment operator or new, are referred to as **unary** operators.
  - Operators that take two operands, such as arithmetic operators, are referred to as **binary operators**.
  - One operator, the conditional operator (?:), takes three operands and is the sole **ternary** operator.

Primary Operators

Expression	Description
x.y	Member access
x?.y	Conditional member access
f(x)	Method and delegate invocation
a[x]	Array and indexer access
a?[x]	Conditional array and indexer access
x++	Post-increment
x--	Post-decrement
new T(...)	Object and delegate creation
`new` T(...){...}	Object creation with initializer. See Object and Collection Initializers.
`new` {...}	Anonymous object initializer. See Anonymous Types.
`new` T[...]	Array creation. See Arrays.
typeof(T)	Obtain System.Type object for T
checked(x)	Evaluate expression in checked context
unchecked(x)	Evaluate expression in unchecked context
default (T)	Obtain default value of type T
delegate {}	Anonymous function (anonymous method)

Unary Operators

Expression	Description
+x	Identity
-x	Negation
!x	Logical negation
~x	Bitwise negation
++x	Pre-increment
--x	Pre-decrement
(T)x	Explicitly convert x to type T

Multiplicative Operators

Expression	Description
*	Multiplication
/	Division
%	Remainder

Additive Operators

Expression	Description
x + y	Addition, string concatenation, delegate combination
x - y	Subtraction, delegate removal

Shift Operators

Expression	Description
x << y	Shift left
x >> y	Shift right

Relational and Type Operators

Expression	Description
x < y	Less than
x > y	Greater than
x <= y	Less than or equal
x >= y	Greater than or equal
x is T	Return true if x is a T, false otherwise
x as T	Return x typed as T, or null if x is not a T

Equality Operators

Expression	Description
x == y	Equal
x != y	Not equal

Logical, Conditional, and Null Operators

Category	Expression	Description
Logical AND	x & y	Integer bitwise AND, Boolean logical AND
Logical XOR	x ^ y	Integer bitwise XOR, Boolean logical XOR
Logical OR	x | y	Integer bitwise OR, Boolean logical OR
Conditional AND	x && y	Evaluates y only if x is true
Conditional OR	x || y	Evaluates y only if x is false
Null coalescing	x ?? y	Evaluates to y if x is null, to x otherwise
Conditional	x ? y : z	Evaluates to y if x is true, z if x is false

Assignment and Anonymous Operators

Expression	Description
=	Assignment
x op= y	Compound assignment. Supports these operators: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
(T x) => y	Anonymous function (lambda expression)