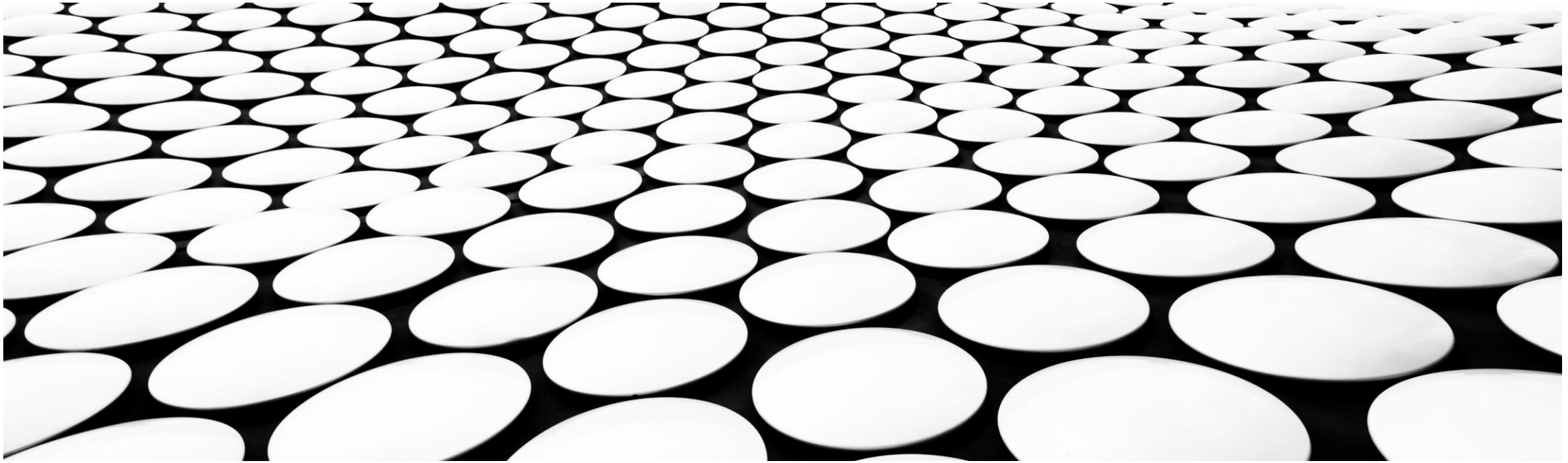
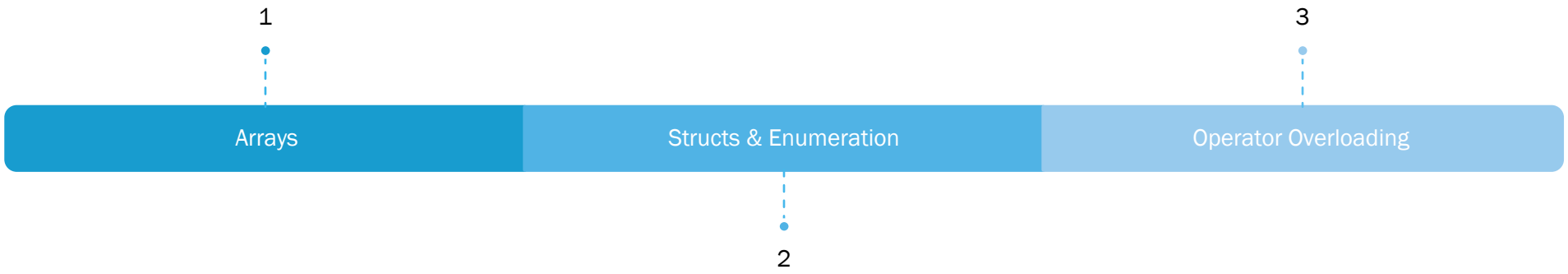
C# - FEATURES

ARCTECH INFO



C# FEATURES



ARRAYS

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with square brackets:
  - string[] cars;
- To declare and initialize an array, use curly braces
  - string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
  - int[] ages = new int[3] { 10, 20, 30 };
  - int[] ages1 = new int[] { 10, 20, 30 };
  - int[] ages2 = { 10, 20, 30 };
  - int[] ages3 = new int[10];
- For late initialization, new is required.

ARRAYS ELEMENTS

- Access array elements
  - Console.WriteLine(cars[0])
- Changes array elements
  - Cars[0] = “Audi”
- Array Length
  - Console.WriteLine(cars.Length);
- Loop Through an Array
  - for and foreach

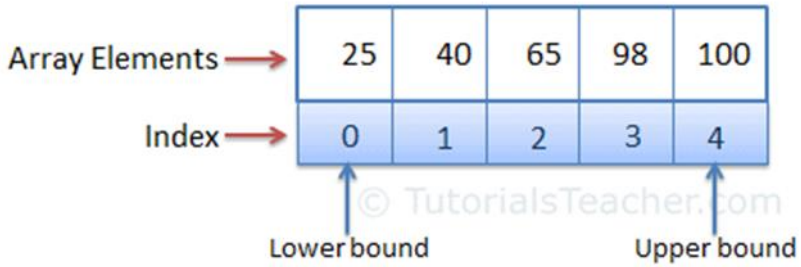
ARRAYS ACTIONS

- System namespace
  - Array.Sort(nums); // sorts array
  - Array.Reverse(nums); // sorts array in descending order
  - Array.ForEach(nums, n => Console.WriteLine(n)); // iterates array
  - Array.BinarySearch(nums, 5);// binary search
- System.Linq namespace
  - Min, Max, and Sum
  - int[] myNumbers = {5, 1, 8, 9};
  - Console.WriteLine(myNumbers.Max()); // returns the largest value

PASSING ARRAYS TO FUNCTIONS

- An array can be passed as an argument to a method parameter. Arrays are reference types, so the method can change the value of the array elements.
- {
  int[] nums = {10, 20, 30};
  UpdateArray(nums);
  }
  public static void UpdateArray(int[] arr)
  {
  …
  }

TYPES OF ARRAYS

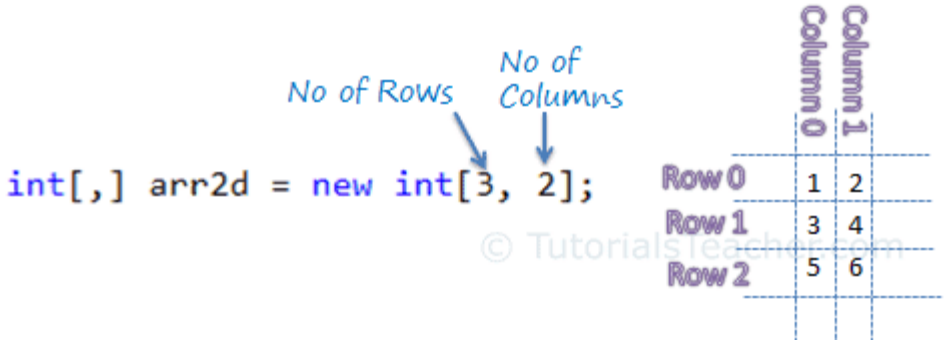


- Single Dimensional
- C# supports multidimensional arrays up to 32 dimensions.
  - The multidimensional array can be declared by adding commas in the square brackets.
  - E.g. [,] declares two-dimensional array,
  - [, ,] declares three-dimensional array, [, , ,] declares four-dimensional array, and so on.
  - So, in a multidimensional array, no of commas = No of Dimensions - 1.
  - `int[,] arr2d; // two-dimensional array`
  - `int[, ,] arr3d; // three-dimensional array`
  - `int[, , ,] arr4d ; // four-dimensional array`
  - `int[, , , ,] arr5d; // five-dimensional array`
- Jagged Arrays

TWO DIMENSIONAL ARRAYS

```
int[,] arr2d = new int[3,2]{
                                {1, 2},
                                {3, 4},
                                {5, 6}
                            };
```

```
// or
int[,] arr2d = {
                        {1, 2},
                        {3, 4},
                        {5, 6}
                    };
```




```
arr2d[0, 0]; //returns 1
arr2d[0, 1]; //returns 2
arr2d[1, 0]; //returns 3
arr2d[1, 1]; //returns 4
arr2d[2, 0]; //returns 5
arr2d[2, 1]; //returns 6
```

3D ARRAYS

```
int[, ,] arr3d1 = new int[1, 2, 2]{
                { { 1, 2}, { 3, 4} }
            };

int[, ,] arr3d2 = new int[2, 2, 2]{
                { {1, 2}, {3, 4} },
                { {5, 6}, {7, 8} }
            };

int[, ,] arr3d3 = new int[2, 2, 3]{
                { { 1, 2, 3}, {4, 5, 6} },
                { { 7, 8, 9}, {10, 11, 12} }
            };
```

```
arr3d2[0, 0, 0]; // returns 1
arr3d2[0, 0, 1]; // returns 2
arr3d2[0, 1, 0]; // returns 3
arr3d2[0, 1, 1]; // returns 4
arr3d2[1, 0, 0]; // returns 5
arr3d2[1, 0, 1]; // returns 6
arr3d2[1, 1, 0]; // returns 7
arr3d2[1, 1, 1]; // returns 8
```

JAGGED ARRAYS

- A jagged array is an array of array.
- Jagged arrays store arrays instead of literal values.
- A jagged array is initialized with two square brackets [][].
- The first bracket specifies the size of an array, and the second bracket specifies the dimensions of the array which is going to be stored.

- `int[][] jArray1 = new int[2][];`
  `// can include two single-dimensional arrays`
- `int[][,] jArray2 = new int[3][,];`
  `// can include three two-dimensional arrays`
- jArray1 can store up to two single-dimensional arrays.
- jArray2 can store up to three two-dimensional, arrays [,] specifies the two-dimensional array.

STRUCTURES AND ENUMERATION

- C# supports two kinds of value types, namely,
  - predefined types
  - userdefined types.
- predefined data types are int, double, etc.
- C# allows us to define our own complex value types
  - known as user defined value types
- There are two kind of value types we can define in C#
  - Structures
  - Enumeration
- Value type variables store their data on the stack.

STRUCTURES

- Structures are similar to classes in C#.
- structs are used when simple composite data types are required.
- Because they are value types, they are stored on the stack,
- Advantages of struct compared to class which are stored on the heap
  - created much more quickly than heap-allocated types.
  - instantly and automatically deallocated once they go out of scope.
  - It is easy to copy value type variables on the stack.
  - The performance of programs may be enhanced by judicious use of structs

STRUCTURE

- Defining a struct

```
struct Student
{
 public string Name;
 public int RollNumber;
 public float TotalMarks;
}
```

- Assigning Values to members

```
s1.Name = "John";
s1.RollNumber = 999 ;
s1.TotalMarks = 575.50 ;
```

- Copying a struct

```
Student s2; // s2 is declared
s2 = s1 ;
```

MORE ABOUT STRUCTS

- A struct can contain methods
- A struct can also contain constructors
  - However, a default constructor is not allowed

OPERATOR OVERLOADING

- What is Overloading?
- Types of Overloading.
- Operator Overloading.
- Types of Operator Overloading.

WHAT IS OVERLOADING?

- Each C# operator has a predefined meaning.
- Most of them are given additional meaning through the concept called operator overloading.
- Main idea behind operator overloading is to use C# operators with class objects.
- It is a form of polymorphism and different to overriding
- Two Types of overloading are
  - 1. Method Overloading
    - Multiple functions with same Name but different types / number of parameters.
  - 2. Operator Overloading.
    - is a way of redefining the meaning of C# operators
    - Done with special functions.

OPERATOR OVERLOADING

- All unary and binary operators have pre-defined implementations
- These are automatically available in any expressions.
- User defined implementations can also be introduced in C#.
- The mechanism of giving a special meaning to a standard C# operator is known as operator overloading.
- Operator overloading is only applicable with respect to a user defined data type such as classes or structures

ALLOWED OPERATORS

Operators	Overloadability
+, -, *, /, %, &, |, <<, >>	All C# binary operators can be overloaded.
+, -, !, ~, ++, --, true, false	All C# unary operators can be overloaded.
==, !=, <, >, <= , >=	All relational operators can be overloaded, but only as pairs.
&&, ||	Can't be overloaded.
[] (Array index operator)	Can't be overloaded.
() (Conversion operator)	Can't be overloaded.
+=, -=, *=, /=, %=	These compound assignment operators can be overloaded. But in C#, these operators are automatically overloaded when the respective binary operator is overloaded.
=, . , ?:, ->, new, is, as, sizeof	Can't be overloaded.

OVERLOADING OPERATORS SYNTAX

- Syntax:

```
public static <ReturnValue> operator <op> (argument list)
{
}
```

- Example:

```
public static Point operator + (Point operator)
{
}
```

- Usage:

```
Point p1;
Point p1 = -p1;
```

TYPES OF OPERATOR OVERLOADING.

- Unary Operator Overloading.
  - You can declare your own version of the increment (++) and decrement (–) operators.
  - • They must be public, static and unary.
  - • They can be used in prefix and postfix forms
- Binary Operator Overloading.
  - At least one parameter must be of the enclosing type.
  - You may overload as many times as you like with different parameter types.
  - You may return any type.
  - “ref” or “out” parameters not allowed

UNARY OPERATOR OVERLOADING

- Unary operator works with single parameter.

```
struct Point
{
	int x;
	int y;
	public static Point operator+(Point p)
	{
		return new Point(p.x, p.y);
	}
	public static Point operator-(Point p)
	{
		return new Point(-p.x, -p.y);
	}
	...
}
```

BINARY OPERATOR OVERLOADING

- works with two parameter.
- Binary operators are of two types
  - Arithmetic/Bitwise Operators
    - + - * / % ^ & | << >>
  - Comparison Operators
    - == != < <= > >=

```
struct Point
{
	int x;
	int y;
	public static Point operator+(Point p, Point q)
	{
		return new Point(p.x + q.x, p.y + q.y);
	}
	public static Point operator-(Point p, Point q)
	{
		return new Point(p.x - q.x, p.y - q.y);
	}
	...
}
```

DETERMINING EQUALITY

- Two kinds of comparison for objects:
- Identity and equality
  - System.Object.Equals method
  - Equality operator(==)
- C# has an "Equals" method which can be used to compare two objects.
- Objects can also be compared using ==

EQUALS

- Even if test1 and test2 contain the same value for FirstName and LastName, the "Equals" method returns False.
- That is because default implementation of Equals method does not check for Equality; it checks for Identity.
- This means test1 and test2 must refer to the exact same object, then only it will return True, otherwise, it will return False.
- As test2 and test3 are referring to the same object, it returns True.
- As per the program we can conclude that default implementation of Equals checks for Identity which means it will return True only if two variables are referring to the same object.

```
class TestEquality
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
}
TestEquality test1 = new TestEquality();
test1.FirstName = "Tom";
test1.LastName = "Cruise";

TestEquality test2 = new TestEquality();
test2.FirstName = "Tom";
test2.LastName = "Cruise";

TestEquality test3 = test2;
bool areEqual = test1.Equals(test2);
Console.WriteLine("Are test1 and test2 are Equal:" + areEqual);

areEqual = test2.Equals(test3);
Console.WriteLine("Are test2 and test3 are Equal:" + areEqual);
```

EQUALS METHOD

- Default implementation of Equals in Object class

```
public virtual bool Equals(Object obj)
{
    //If both the references points to the same object then only return true
    if (this == obj)
    {
        return true;
    }
    return false;
}
```

- We can override the Equals method

== & != OPERATOR

- The default implementation returns the result of comparing the two references for equality or non-equality.
- Since the predefined reference type equality operators accept operands of type object, they apply to all types that do not declare applicable operator == and operator != members.
- We can overload the == operator

ADVANTAGES OF OPERATOR OVERLOADING

- readability of the code improves.
- the code becomes explicit in nature,
- looking at the code, fellow developers can easily guess what is going on.
- With operator overloading the code looks more conventional and becomes easy to follow.
- The method defining operator overloading, should always be static and public.
  - Otherwise, compiler errors out with message