C# Learning Notes (Easy-to-Understand Guide)

1. Basic Syntax and Variables

- **Data Types**: Used to specify the type of data stored.
- Integers (`int`): Whole numbers, e.g., `int count = 5;`
- Double (`double`): Decimal numbers, e.g., `double price = 19.99;`
- String (`string`): Text, e.g., `string name = "Alice";`
- Boolean (`bool`): True or False, e.g., `bool isActive = true;`
- Decimal (`decimal`): Precise decimal numbers, e.g., `decimal salary = 50000.00m;`
- **Constants**: Values that do not change once set.

```
const int MAX_USERS = 100;
```

2. Control Flow

- **Conditional Statements**: Decide which code to execute based on conditions.

```
if (score > 80) {
  Console.WriteLine("Great job!");
} else if (score > 60) {
  Console.WriteLine("Good effort.");
} else {
  Console.WriteLine("Keep trying!");
}
```

```
switch(day)
{
  case "Monday":
    Console.WriteLine("Start of the week!");
    break;
  default:
    Console.WriteLine("Another day!");
    break;
}
```

- **Loops**: Repeat tasks multiple times.

```
for(int i = 0; i < 10; i++) {
  Console.WriteLine(i);
}
```

```
int count = 0;
while(count < 5) {
  Console.WriteLine(count);
  count++;
}
```

3. Functions and Methods

- **Reusable blocks of code** to simplify complex tasks.

```
int Multiply(int x, int y) {
  return x * y;
}

int result = Multiply(3, 4); // result is 12
```

4. Object-Oriented Programming (OOP)

- **Classes & Objects**: Define blueprints and create instances.

```
class Animal
{
  public string Name { get; set; }

  public Animal(string name) {
    Name = name;
  }

  public void Speak() {
    Console.WriteLine($"{Name} makes a sound.");
  }
}

Animal dog = new Animal("Buddy");
dog.Speak();
```

- **Inheritance & Polymorphism**: Create specialized classes and methods.

```
class Dog : Animal
{
  public Dog(string name) : base(name) {}

  public override void Speak() {
    Console.WriteLine($"{Name} barks.");
  }
}
```

- **Encapsulation**: Control data access and modification.

```
private int age;
public int Age
{
  get { return age; }
  set { if(value >= 0) age = value; }
}
```

5. Collections

- **Array**: Fixed-size list of similar items.

```
int[] scores = {80, 90, 100};
```

- **List**: Flexible-sized collections.

```
List<string> names = new List<string>() {"Anna", "Bob"};
```

- **Dictionary**: Key-value pairs for quick access.

```
Dictionary<int, string> students = new Dictionary<int, string>();
students.Add(1, "Alice");
```

6. LINQ (Language Integrated Query)

- Easily filter and sort data.

```
var evenNumbers = scores.Where(n => n % 2 == 0);
var sortedScores = scores.OrderByDescending(n => n);
```

7. Exception Handling

- Safely manage unexpected errors.

```
try
{
  int number = int.Parse(input);
}
catch(FormatException ex)
{
  Console.WriteLine("Please enter a valid number.");
}
finally
{
  Console.WriteLine("Finished checking input.");
}
```

Helpful Tips

- Clearly name variables and methods to enhance readability.
- Regularly test your code with different inputs to catch errors early.
- Use comments to clarify complex logic or decisions.

This guide simplifies essential C# concepts clearly to strengthen understanding and memory retention.