데이터 형식 (Data Type)

C#에서 제공하는 데이터 형식은 숫자, 텍스트 뿐만 아니라, 이미지, 소리까지 다룰 수 있는 데이터 형식을 제공합니다.
데이터 형식은 기본 데이터형식(Primitive Type) 과 복합 데이터형식 (Complex Data Type) 으로 나눌 수 있습니다.
기본 데이터형식은 흔히 생각하는 int , double 과 같은 개념이라 생각하면 되고,
복합 데이터형식은 기본데이터 형식을 기본으로 하는 구조체나 클래스 등이 되겠습니다.
기본 데이터형식과 복합 데이터형식을 다시 값 형식과 참조 형식으로 분류할 수 있습니다.

값 형식 (Value Types) 과 참조 형식 (Reference Types)

값 형식은 메모리 영역 중 스택(Stack) 영역 에 저장되며, 선언 되었던 코드 블록이 끝나면 스택영역에서 데이터가 제거됩니다.
스택에 쌓인 데이터들이 코드 블록이 끝나는 순간 제거 된다는 것은 값 형식의 장점이자 단점이 됩니다.
코드 블록이 끝나는 순간 데이터가 제거되기 때문에 메모리영역이 전부 회수되지만, 코드 블록안에서만 사용가능합니다.
참조 형식은 메모리 영역 중 힙(Heap) 영역에 데이터를 저장하는데, 힙 영역은 코드 블록과 상관 없이 데이터는 사라지지 않습니다. 참조 형식의 변수는 힙과 스택을 동시에 이용합니다. 힙 영역에는 데이터의 값을 저장하고, 스택 영역에는 이 데이터의 주소를 저장합니다. 스택 영역은 코드 블록이 끝나는 순간 사라지지만 (데이터의 주소), 힙 영역에 데이터의 값은 사라지지 않습니다. C# 에서는 CLR의 가비지 컬렉터 (Garbage Collector) 가 힙에 사용하지 않는 객체를 자동으로 제거해줍니다.

object 형식

C#은 모든 데이터 형식을 다룰 수 있는 object형식이 존재합니다. 모든 데이터 형식 (기본 데이터 형식 뿐만 아니라 모든 복합 데이터 형식, 프로그래머가 만드는 데이터형식 까지도) 자동으로 object 형식으로부터 상속받게 하였기 때문입니다.
정수 형식은 short 와 int 의 처리가 다르고, 부동 소수점 형식은 float 와 double 가 처리가 다른데도 가능한 이유는 박싱(boxing) 과 언박싱(unboxing) 이라는 메커니즘이 있기 때문입니다.

박싱(boxing) 과 언박싱(unboxing)

object 형식은 참조 형식이기 때문에 힙에 데이터를 할당합니다.
하지만 object 형식에 값 형식의 데이터 (int와 같은)를 할당하면 object 형식은 박싱을 수행해서 데이터를 힙에 할당합니다. 이 object 형식을 다시 값 형식의 변수에 할당하면, object 형식의 데이터를 언박싱하여 데이터 값을 값 형식의 변수에 할당합니다.
예를 들어, 다음과 같은 코드가 있습니다.

```
object a= 10;
int b = (int) a;
```

이럴 경우 10을 박싱하여 힙에 저장하고, a는 박싱한 10의 주소를 갖습니다. 다시 b에 저장하고자 할때 a는 언박싱되어 10이 b에 저장됩니다.

열거 형식(Enumerator)
종류는 같지만, 다른 값을 갖는 상수들을 선언해야 할때 유용합니다.
예를 들면, 메시지 박스를 띄웠을 때 사용자로 부터 받는 응답이 Yes, No, Cancel 과 같을 때 상수로 선언하여 컨트롤하면 쉽습니다.
enum 열거형식명 : 기반자료형 { 상수1, 상수2, ... }
기반 자료형은 정수 계열(byte, short, int, long 등)만 사용할 수 있습니다. 생략할 경우 컴파일러가 int 형으로 사용합니다.

enum 의 여러가지 사용방법은 다음과 같습니다.

```
using System; using System.Collections.Generic;
 using System.Linq;
using System.Text;
 using System.Threading.Tasks;
 namespace CsharpStudy
 {
class Program
{
enum DialogResult
 {
 YES=1 ,
NO ,
 CANCEL
}
static void Main(string[] args)
 {
Console.WriteLine(DialogResult.YES);
 Console.WriteLine(DialogResult.NO);
 Console.WriteLine(DialogResult.CANCEL);
 Console.WriteLine(); Console.WriteLine((int)DialogResult.YES);
 Console.WriteLine((int)DialogResult.NO);
Console.WriteLine((int)DialogResult.CANCEL);
 Console.WriteLine();
 Console.WriteLine( (DialogResult) 1);
Console.WriteLine( (DialogResult) 2);
 Console.WriteLine( (DialogResult) 3);
 }
 }
 }
```

```
YES
NO
CANCEL

1
2
3

YES
NO
CANCEL
```

Nullable 형식
C#에서는 메모리 공간에 어떤 값이든 넣도록 강제합니다. 하지만 어떤 값도 가지지 않는 변수가 필요할 때가 있습니다. 0 이 아니라 아예 비어있는 변수, 즉 null 한 변수를 말합니다.
데이터형식 ? 변수이름;

```
using System;
using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
 namespace CsharpStudy
 {
 class Program
 {
 static void Main(string[] args)
 {
 int? a = null;
double? b = null;
 Console.WriteLine(a.HasValue);
Console.WriteLine(a == b); a = 3;
Console.WriteLine();
 Console.WriteLine(a.HasValue);
Console.WriteLine(a != null);
 Console.WriteLine(a.Value);
}
 }
}
```

```
False
True

True
True
3
```

var 형식

C#은 변수나 상수에 대해 깐깐하게 형식 검사를 하는 강력한 형식의 언어 (Strong Typed Language) 입니다. 이는 프로그래머의 실수를 줄여줍니다. 하지만 int, uint, long, ulong 등 수많은 형식을 외워야 하기 때문에 단점도 존재합니다. 이를 극복하기 위해 var 키워드를 통해 컴파일러가 해당 변수의 형식을 알아서 지정해줍니다.

```
var a= 3;  // a는 int형식
var b= "Hello"; // b는 string 형식
```

var 형식은 지역 변수로만 사용할 수 있습니다. 그리고 클래스의 필드를 선언할 때는 반드시 명시적인 형식을 선언해야 합니다.
클래스의 필드는 보통 생성자에서 초기화하게 되는데 var키워드로 선언하면 무슨 형식인지 컴파일러가 알 수 없기 때문입니다.
참고로 C#에서는 전역변수를 지원하지 않습니다. 코드의 가독성을 해치고 오류를 낳는 원흉으로 지적되었기 때문에 전역변수를 지원하지 않게 되었습니다.

다음과 같이 선언하면 var 와 object 가 헷갈릴 수 있으나 전혀 다른 개념입니다.

```
object a= 10;
var b = 10;
```

위 코드가 컴파일되어 실행하면 object 형식에서 CLR은 20을 박싱해서 힙에 넣고 a가 힙을 가리키게 됩니다.
var 형식에서 컴파일 시점에 컴파일러가 적합한 데이터형식을 파악하여 int b=10; 으로 되어 CLR은 int b=10; 으로 스택에 10을 올립니다.

이상으로 C#에서 데이터 다루는 방식을 정리보았습니다.