

# AI기초실습

# 강의 소개

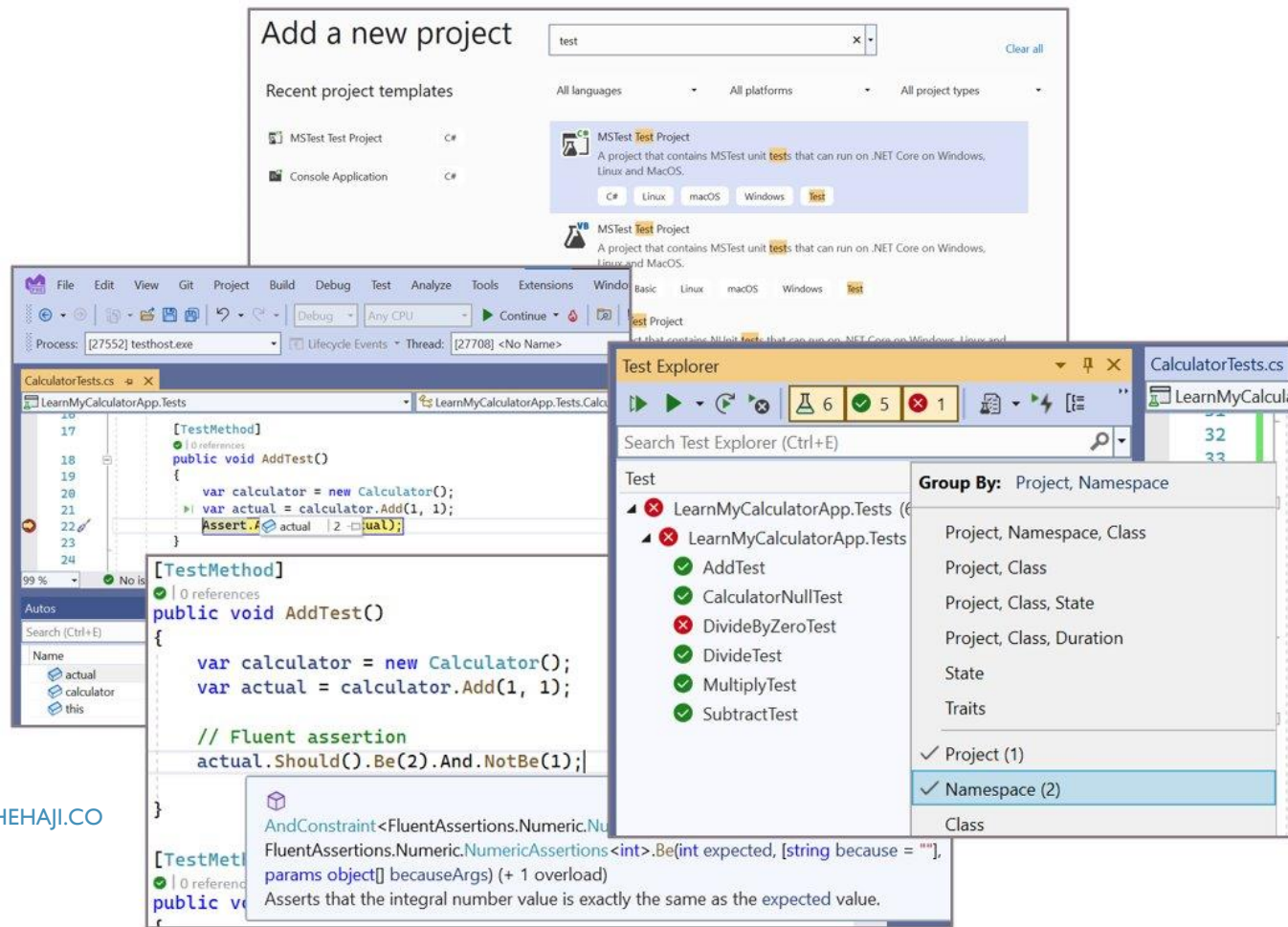
1. C# 언어
2. 기초 문법 입문 & 예제 실습
3. 객체지향 언어 & 클래스
4. 람다, 비동기 호출
5. GUI 프로그래밍 기초
6. 기초 알고리즘 (리스트, 맵, 큐, 스택)
7. 쓰레드, 파일 입출력
8. 네트워크 통신
9. 데이터 베이스 기초 & 실습
10. 디자인 패턴

## # 참고 도서



소개

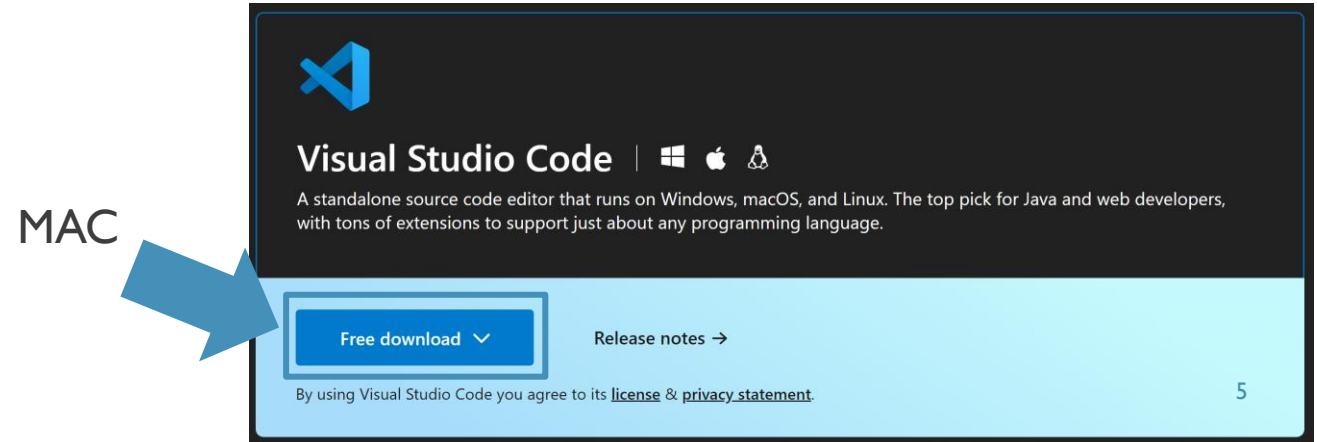
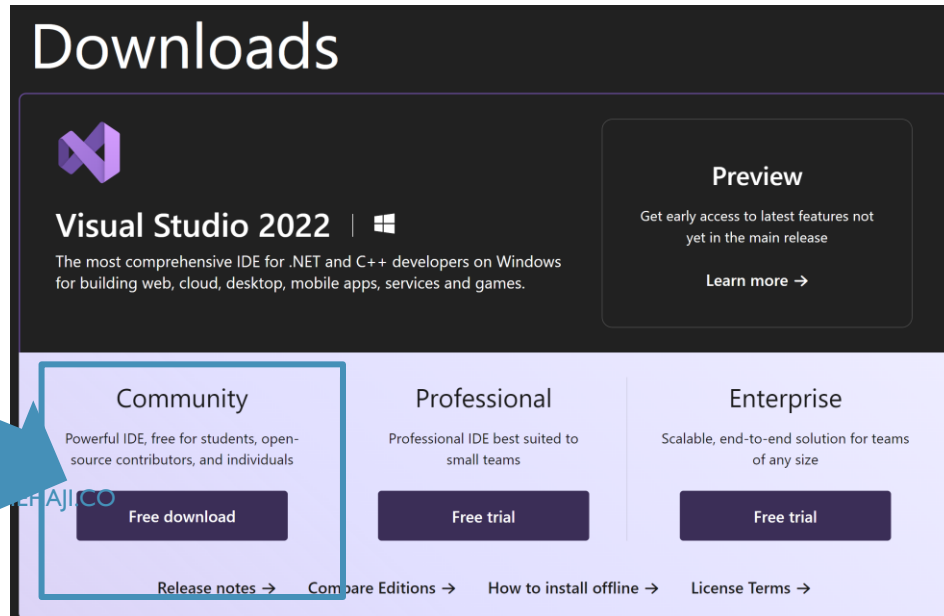
# C# 개발환경 설치



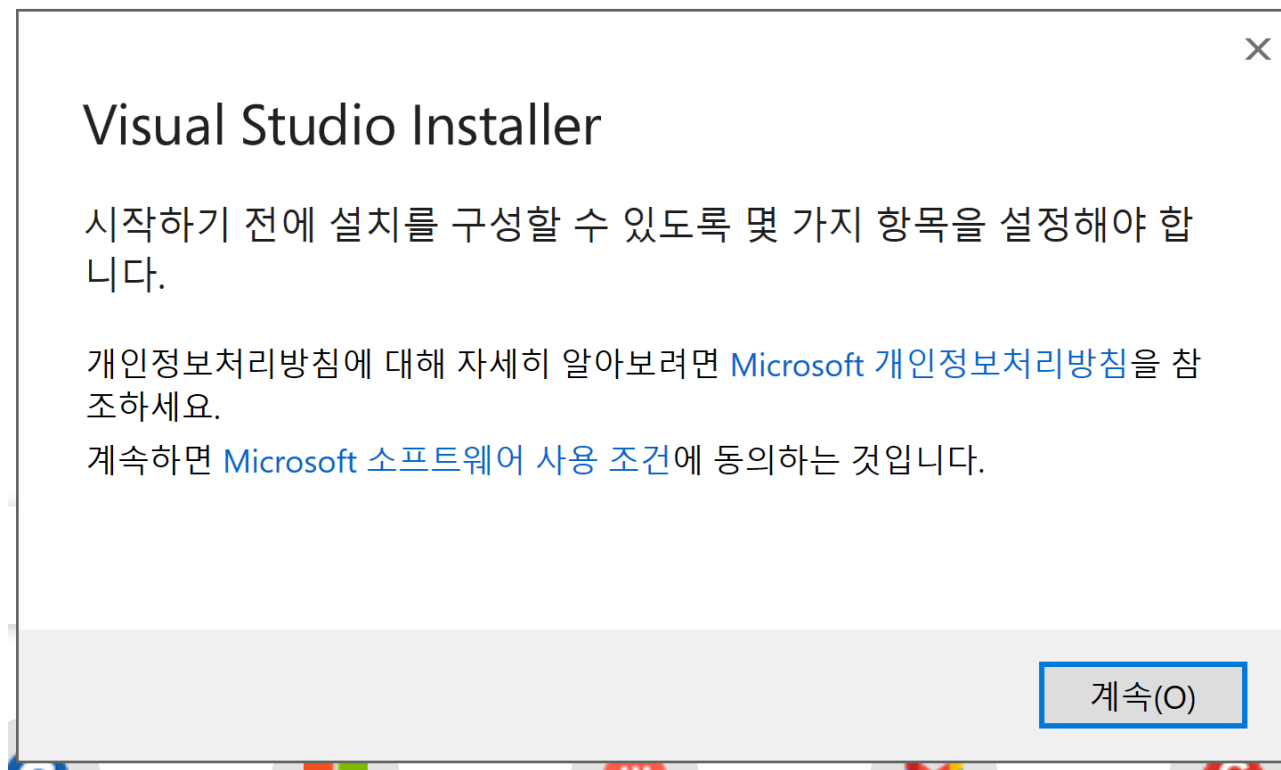
- MAC :VS Code용 새 C# Dev Kit 확장  
<https://learn.microsoft.com/ko-kr/visualstudio/mac/installation?view=vsmac-2022>  
<https://code.visualstudio.com/docs/csharp/get-started>
- MS :Visual Studio  
<https://learn.microsoft.com/ko-kr/visualstudio/install/install-visual-studio?view=vs-2022>
- Visualstudio 다운로드 : <https://visualstudio.microsoft.com/downloads/>

Mac용 Visual Studio Microsoft  
에 따라 2024년 8월 31일에 사용  
중지

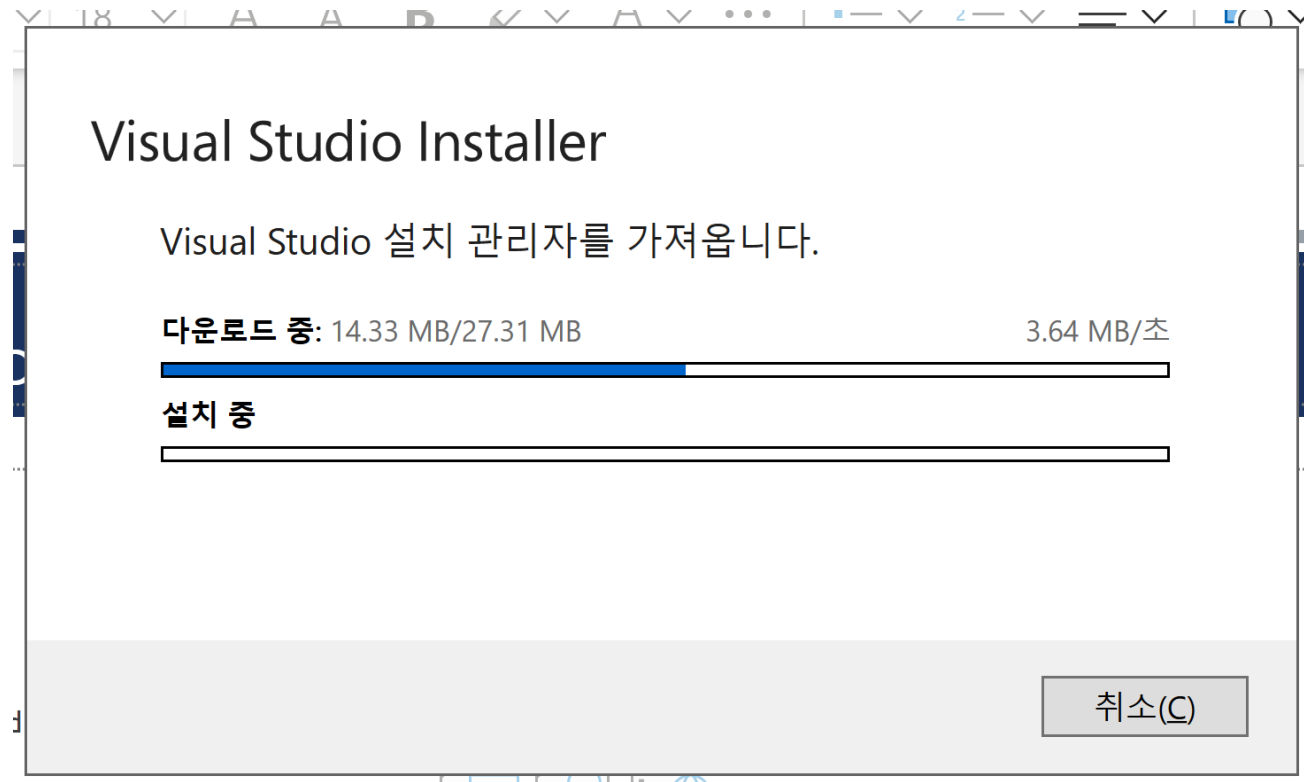
유료 버전만 지원했던 비주얼  
스튜디오가 2013년 부터는 무료  
버전인 비주얼 스튜디오  
커뮤니티도 지원하고 있습니다.



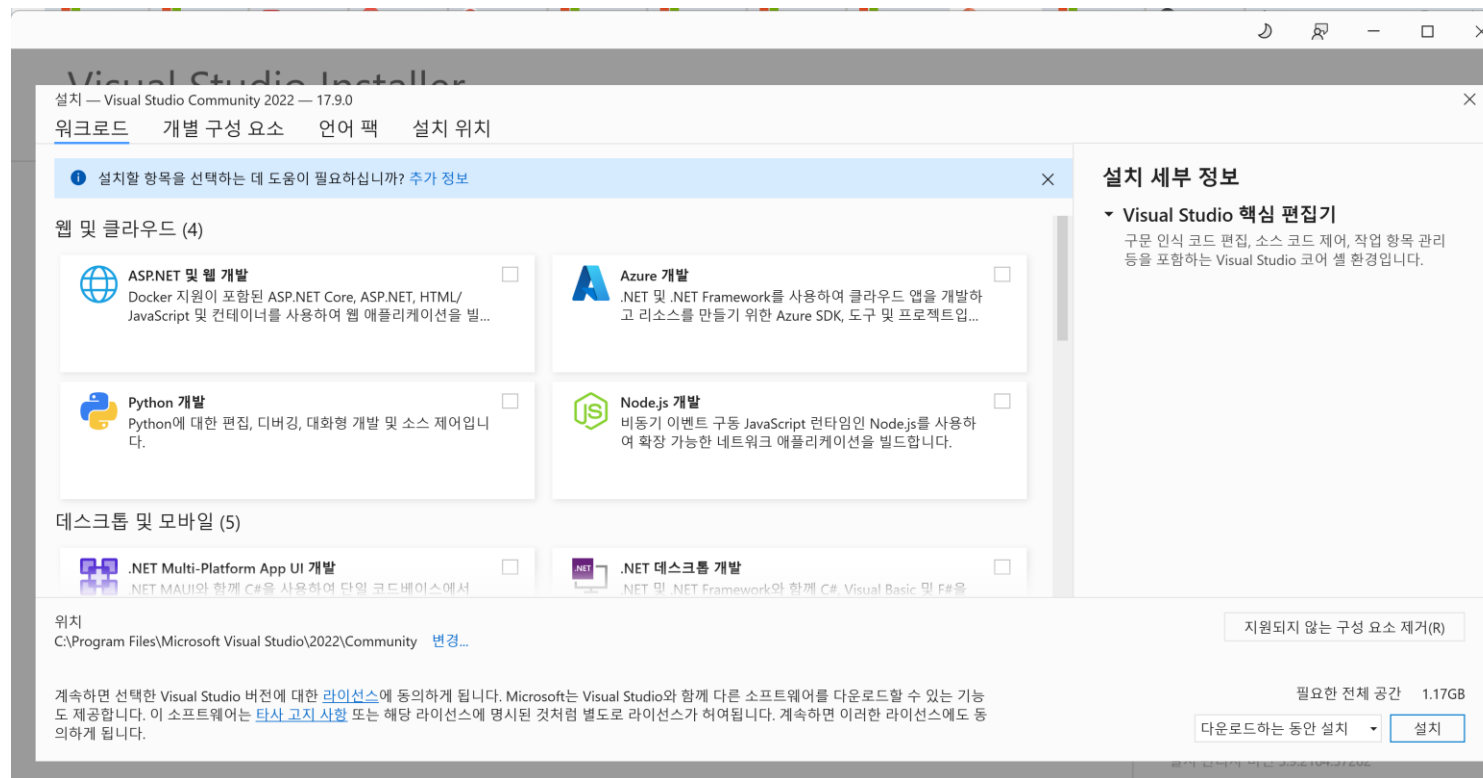
# C# 개발환경 설치(MS)



# C# 개발환경 설치(MS)

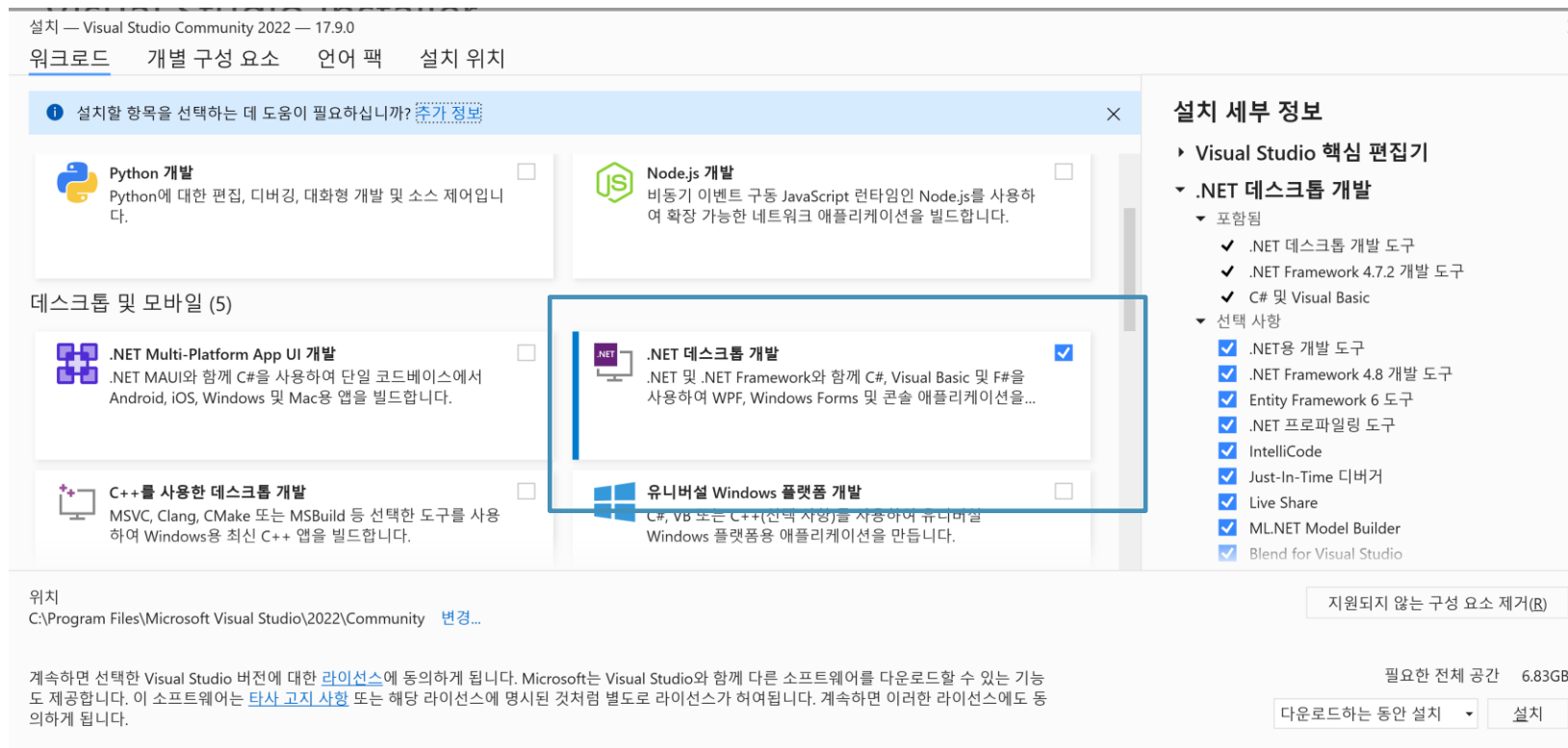


# C# 개발환경 설치(MS)





# C# 개발환경 설치(MS)



# C# 개발환경 설치(MS)

Visual Studio 초보자

설치

시작

지금 자습서 시작

C++

**.NET**

JavaScript/TypeScript

Node.js

Python

영역별로 스킬 확장

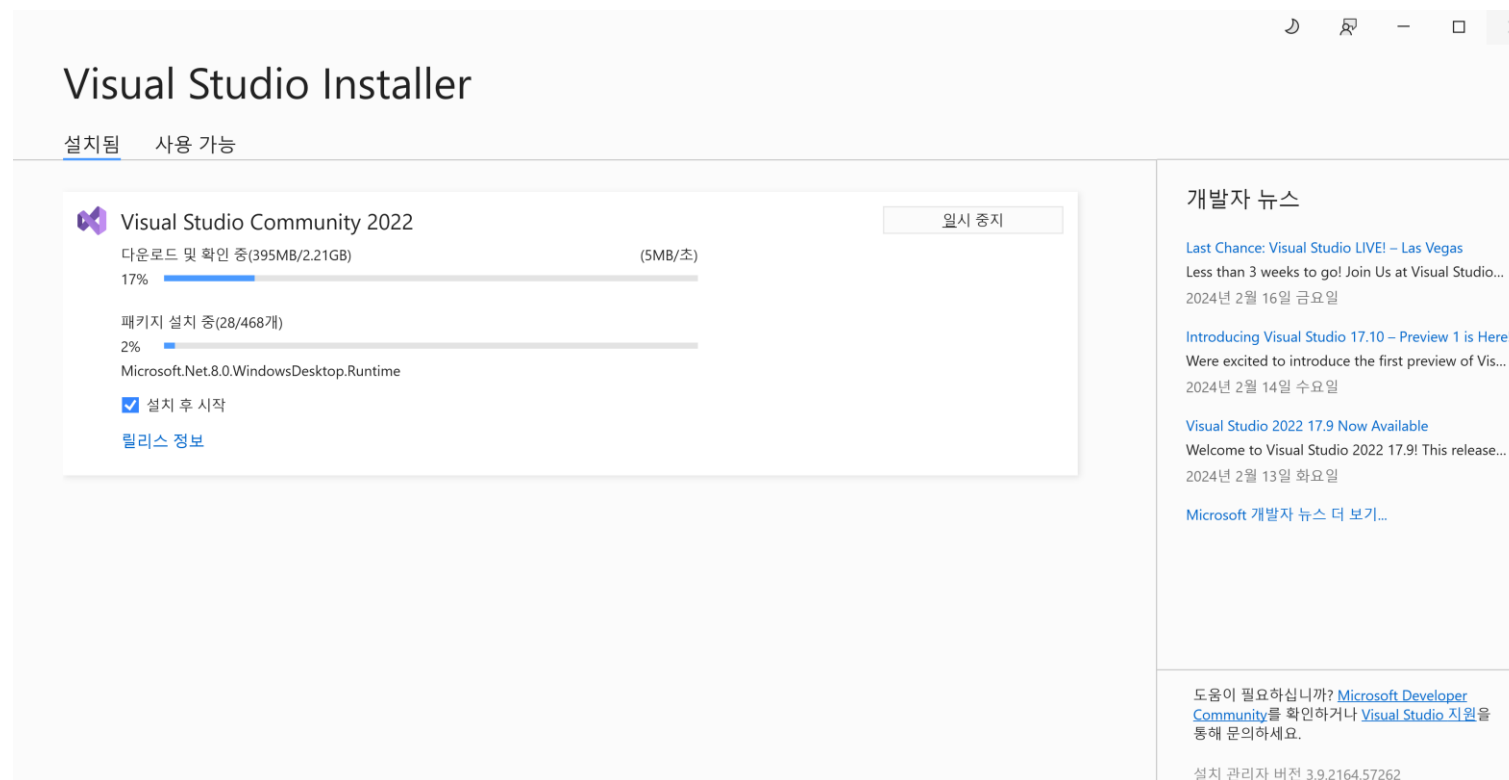
**다른 워크로드 선택시 참고**

**.NET**

**.NET**

Visual Studio 및 .NET을 사용하여 데스크톱, 웹, 모바일, 게임 및 IoT용 응용 프로그램을 개발할 수 있습니다. C#, F# 또는 Visual Basic 언어로 .NET 앱을 작성할 수 있습니다.

# C# 개발환경 설치(MS)



# C# 개발환경 설치(MS)



## Visual Studio 환경 개인 설정

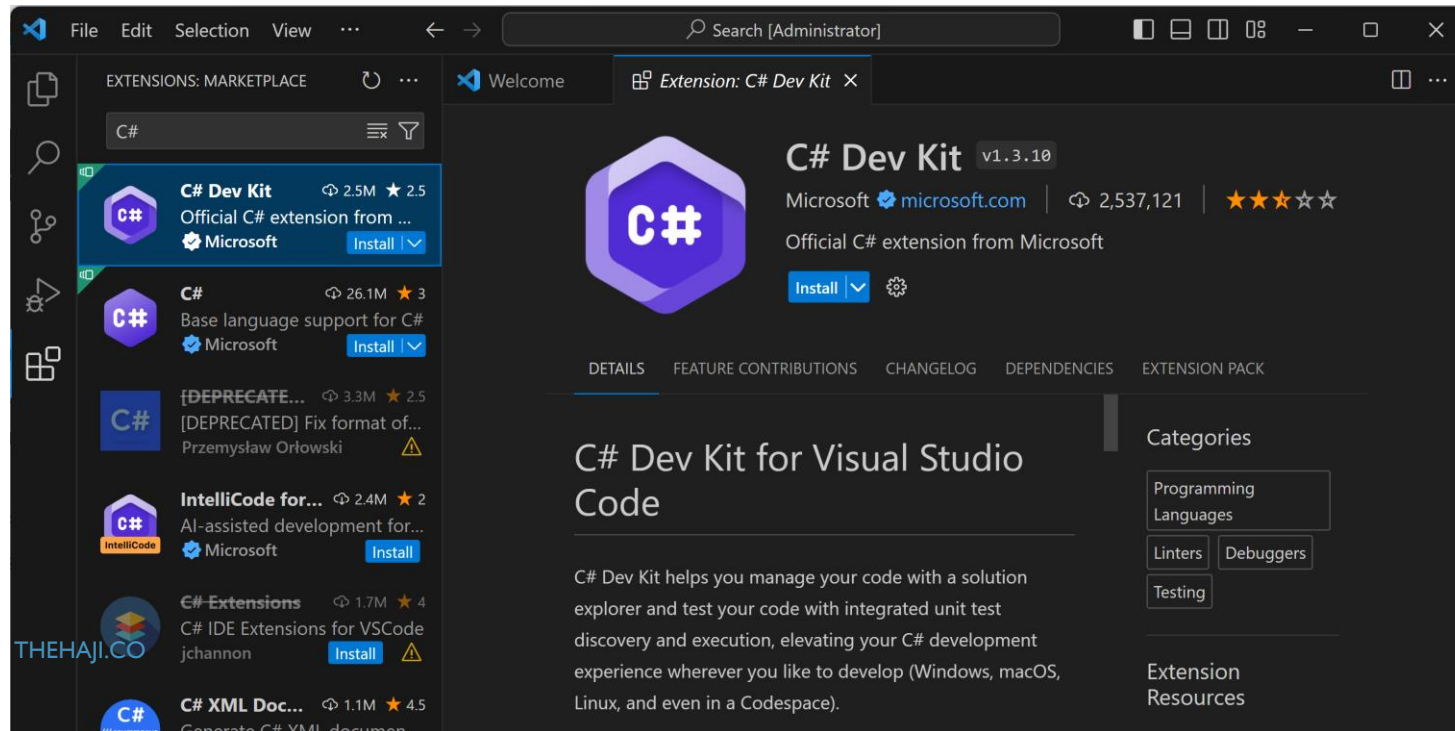
처음 사용하도록 설정하는 동안 잠시 기다려 주세요

이 작업은 몇 분 정도 걸릴 수 있습니다...

• • • • •

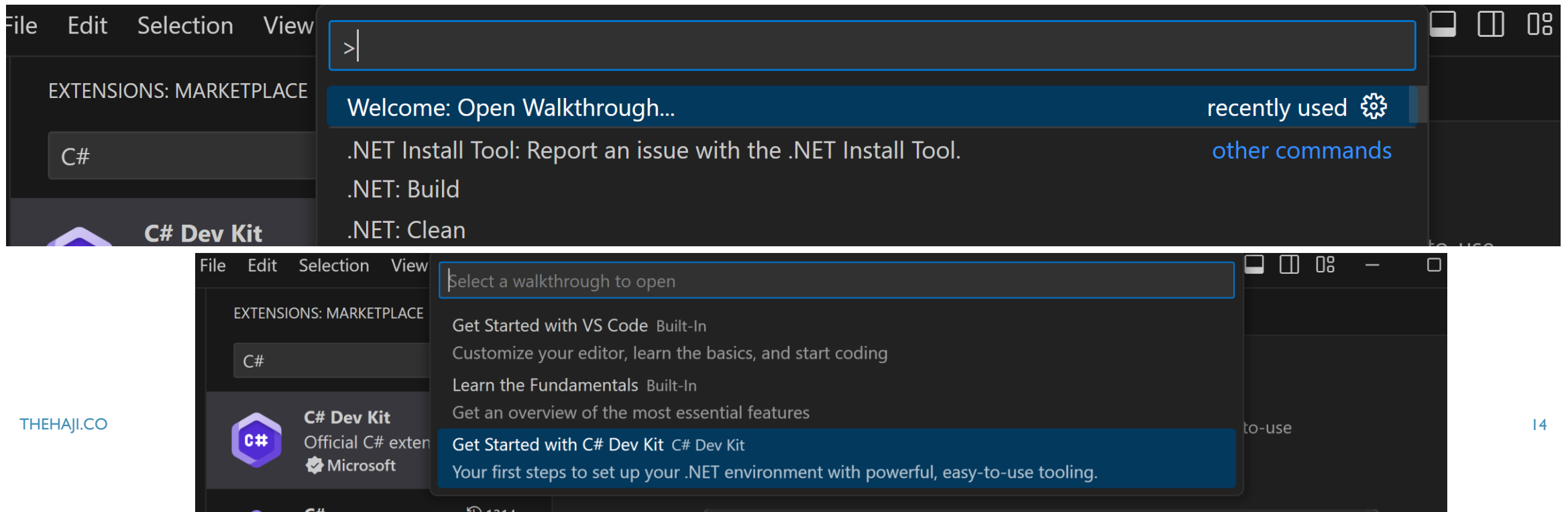
# C# 개발환경 설치(MAC)

1. install VS Code.
2. Next, install C# Dev Kit from the Visual Studio Marketplace.

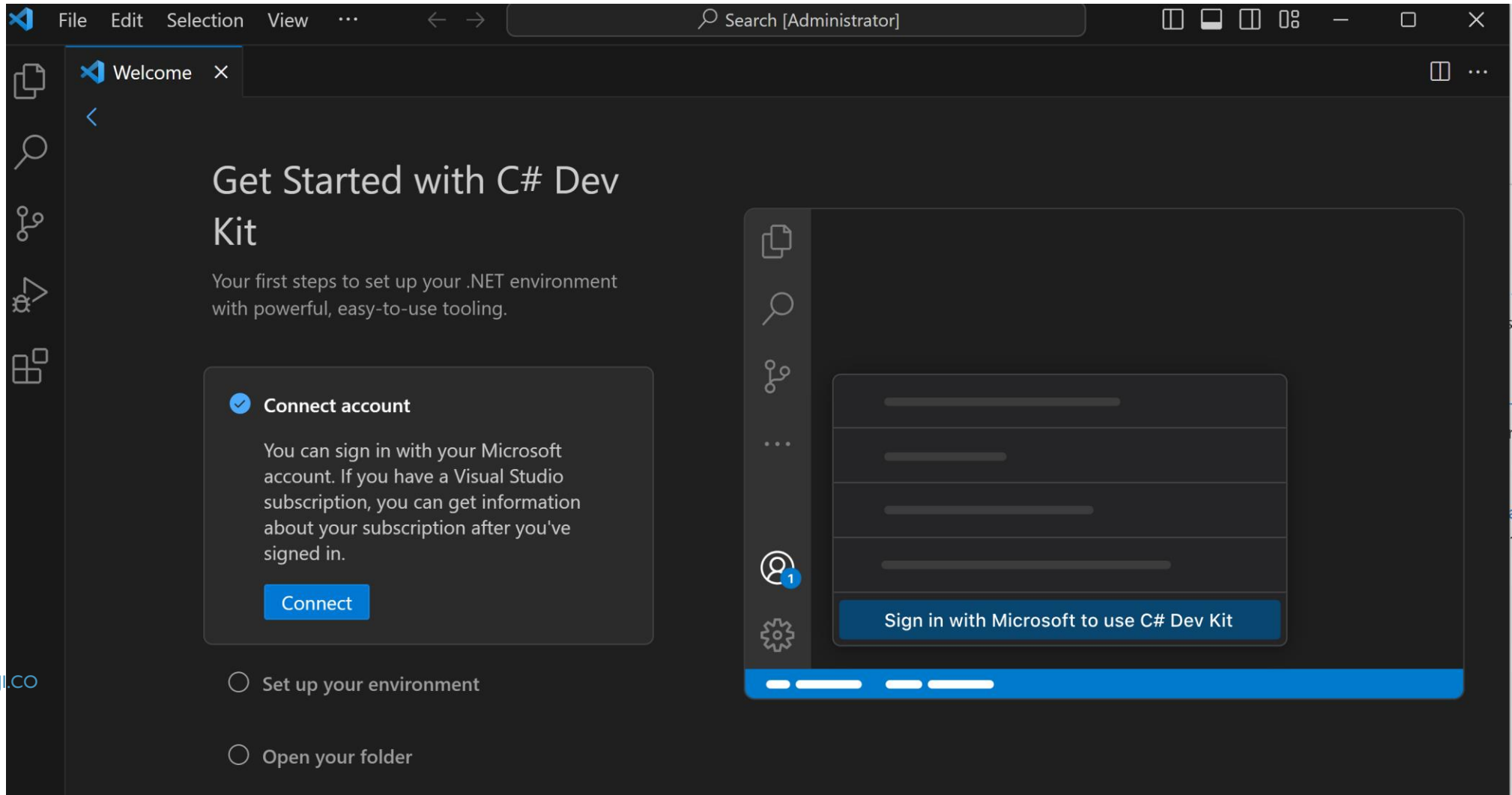


# C# 개발환경 설치(MAC)

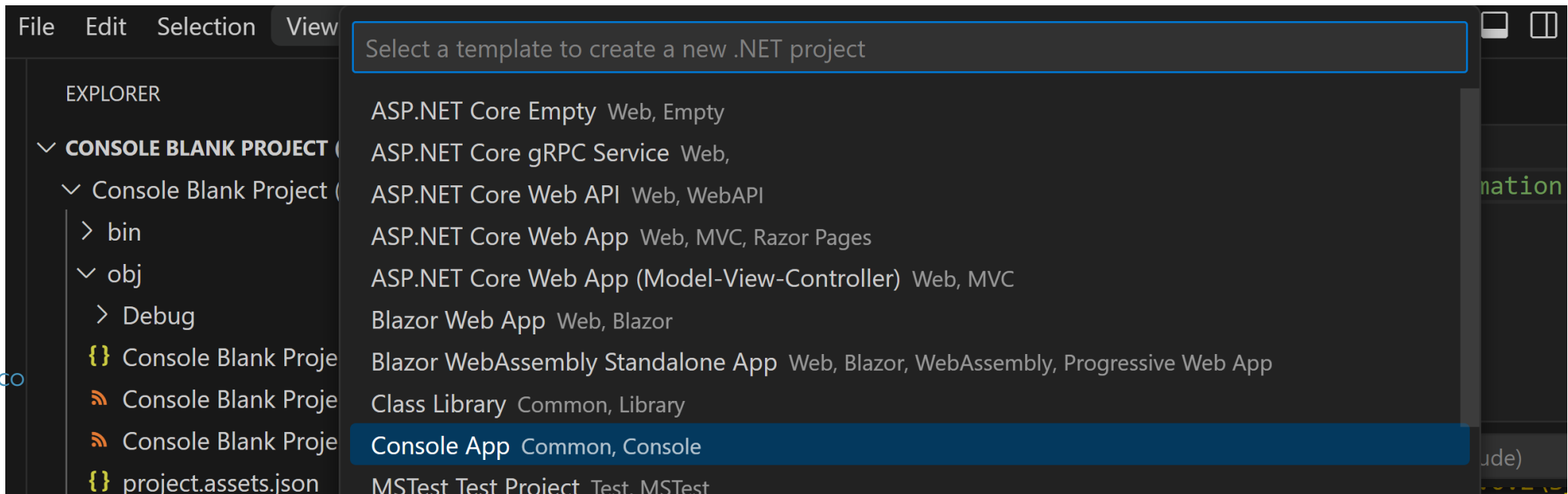
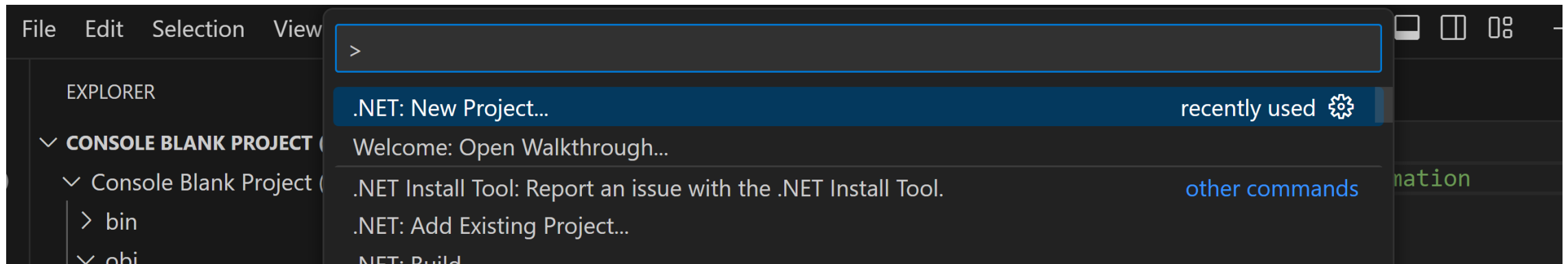
1. Command Palette (**Ctrl+Shift+P**)
2. selecting **Welcome: Open Walkthrough.**
3. select **Get Started with C# Dev Kit.**



# C# 개발환경 설치(MAC)

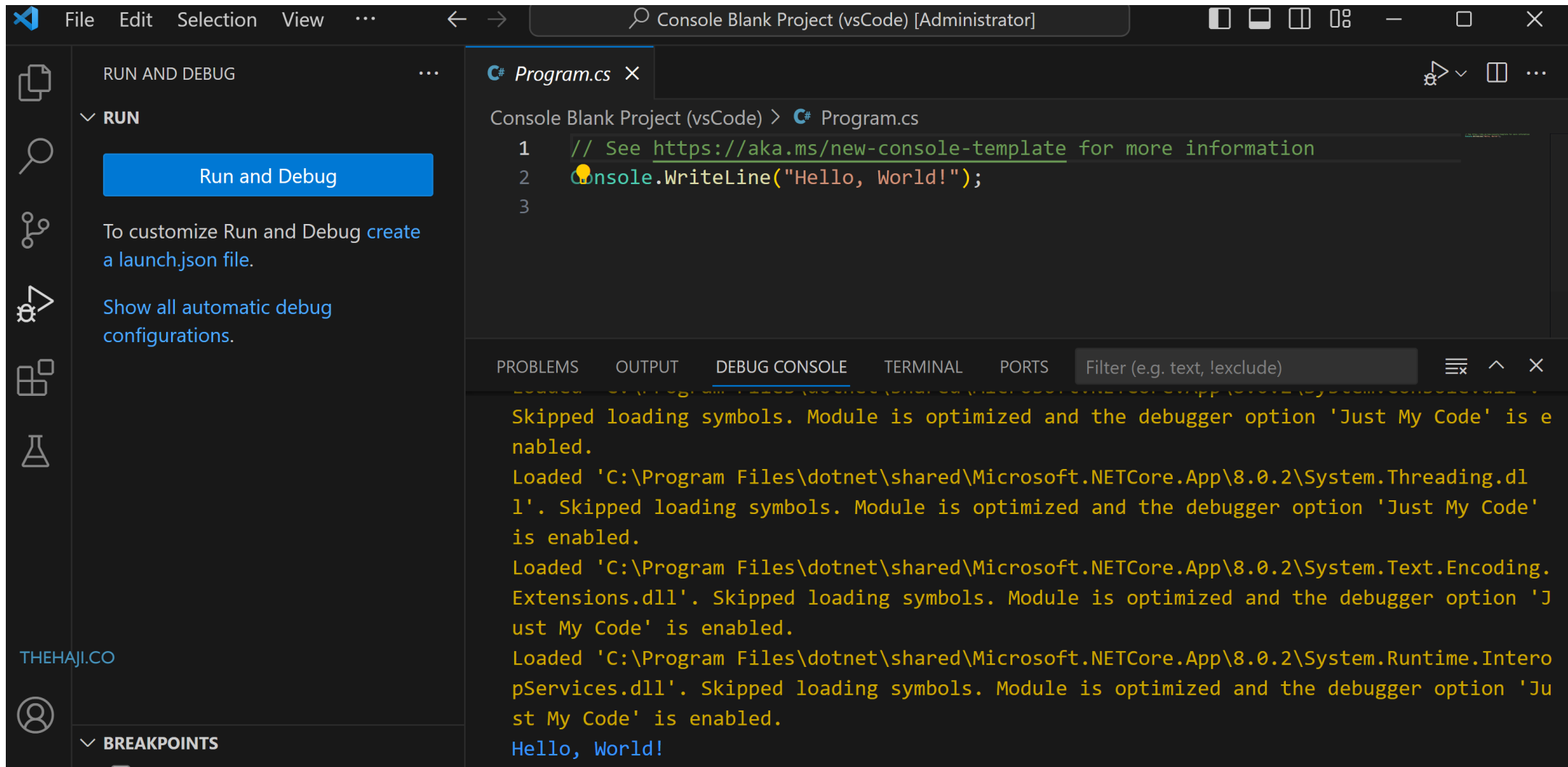


# C# 개발환경 설치(MAC)





# C# 개발환경 설치(MAC)




# 새 프로젝트 생성





# 새 프로젝트 생성

## 새 프로젝트 만들기

### 최근 프로젝트 템플릿(R)

 Windows Forms 앱

 콘솔 앱

 빈 프로젝트(.NET Framework)

C#

C#

C#

템플릿 검색(Alt+S)(S)



모두 지우기(C)

C#

모든 플랫폼(P)

모든 프로젝트 형식(T)



콘솔 앱

Windows, Linux 및 macOS의 .NET에서 실행할 수 있는 명령줄 응용 프로그램을 만들기 위한 프로젝트

C#

Linux

macOS

Windows

콘솔



클래스 라이브러리

.NET 또는 .NET Standard를 대상으로 하는 클래스 라이브러리를 만들기 위한 프로젝트

C#

Android

Linux

macOS

Windows

라이브러리



Blazor 서버 앱이 비어 있습니다.

ASP.NET Core 앱 내에서 서버 쪽을 실행하고 SignalR 연결을 통해 사용자 상호 작용을 처리하는 Blazor 서버 앱을 만들기 위한 빈 프로젝트 템플릿입니다. 이 템플릿

# 새 프로젝트 생성

HelloWorld

새 프로젝트 구성

콘솔 앱 C# Linux macOS Windows 콘솔

프로젝트 이름(I)

helloworld

위치(L)

C:\Project\WC# project

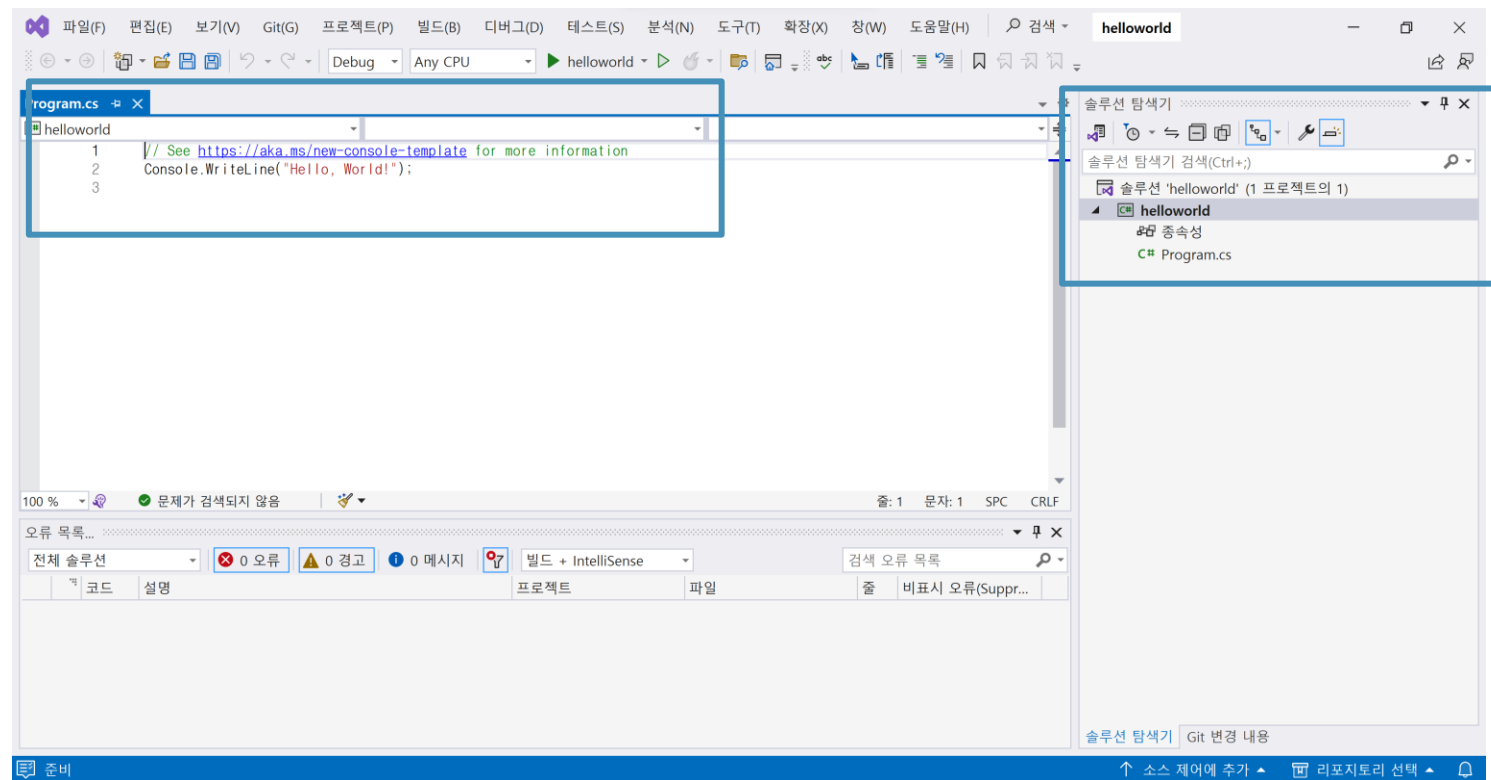
솔루션 이름(M) ⓘ

helloworld

☐ 솔루션 및 프로젝트를 같은 디렉터리에 배치(D)

"C:\Project\WC# project\helloworld\helloworld\W"에 프로젝트이(가) 만들어집니다.

# 새 프로젝트 생성



Create a new project

Save your project

Run your app

Solution Explorer

Sign in

Use Git to  
manage  
code

Set  
breakpoints

The screenshot displays the Visual Studio IDE with a C# project named 'Hello World'. The main editor window shows the code for 'MainPage.xaml.cs'. The code includes using statements for various namespaces and a partial class 'MainPage' that inherits from 'Page'. The code is as follows:

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Runtime.InteropServices.WindowsRuntime;
6 using Windows.Foundation;
7 using Windows.Foundation.Collections;
8 using Windows.UI.Xaml;
9 using Windows.UI.Xaml.Controls;
10 using Windows.UI.Xaml.Controls.Primitives;
11 using Windows.UI.Xaml.Data;
12 using Windows.UI.Xaml.Input;
13 using Windows.UI.Xaml.Media;
14 using Windows.UI.Xaml.Navigation;
15
16 // The Blank Page item template is documented at https://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409
17
18 namespace Hello_World
19 {
20     /// <summary>
21     /// An empty page that can be used on its own or navigated to within a Frame.
22     /// </summary>
23     public sealed partial class MainPage : Page
24     {
25         public MainPage()
26         {
27             this.InitializeComponent();
28         }
29     }
30 }
31
```

The Solution Explorer on the right shows the project structure, including 'App.xaml', 'MainPage.xaml', 'MainPage.xaml.cs', and 'Package.appxmanifest'. The Team Explorer at the bottom right shows the 'Connect' button and 'Hosted Service Providers' section, including 'Azure DevOps'. The Output Window at the bottom shows the debug output, including the message 'The program '[16812] Hello World.exe' has exited with code 1 (0x1)'.

View alerts

Output Window

Editor Window

Team Explorer

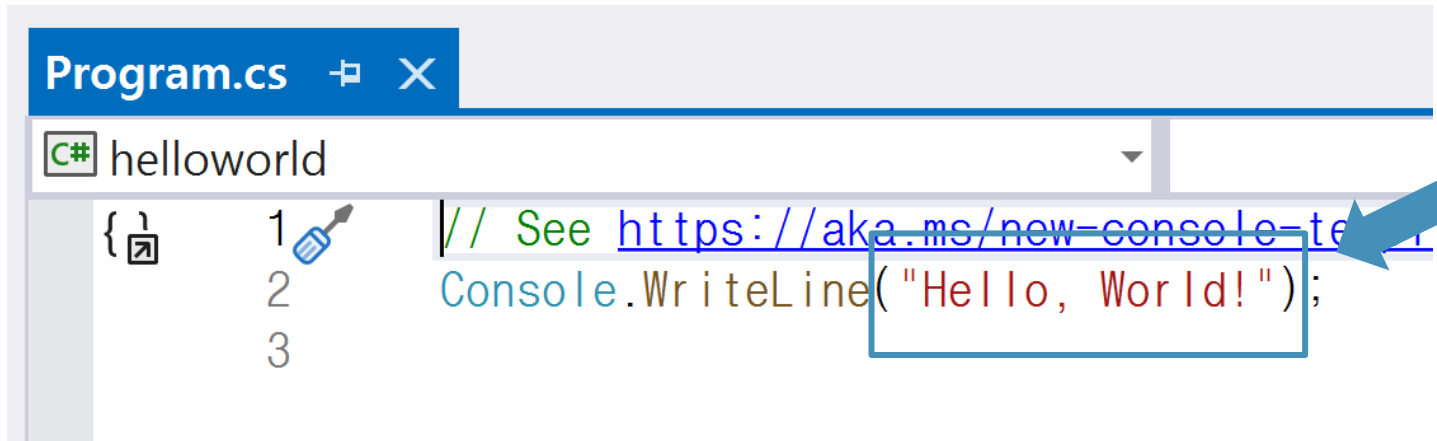
## 새 프로젝트 생성



[://aka.ms/new-console-template](https://aka.ms/new-console-template) for more information

```
Line( "Hello, World! " );
```

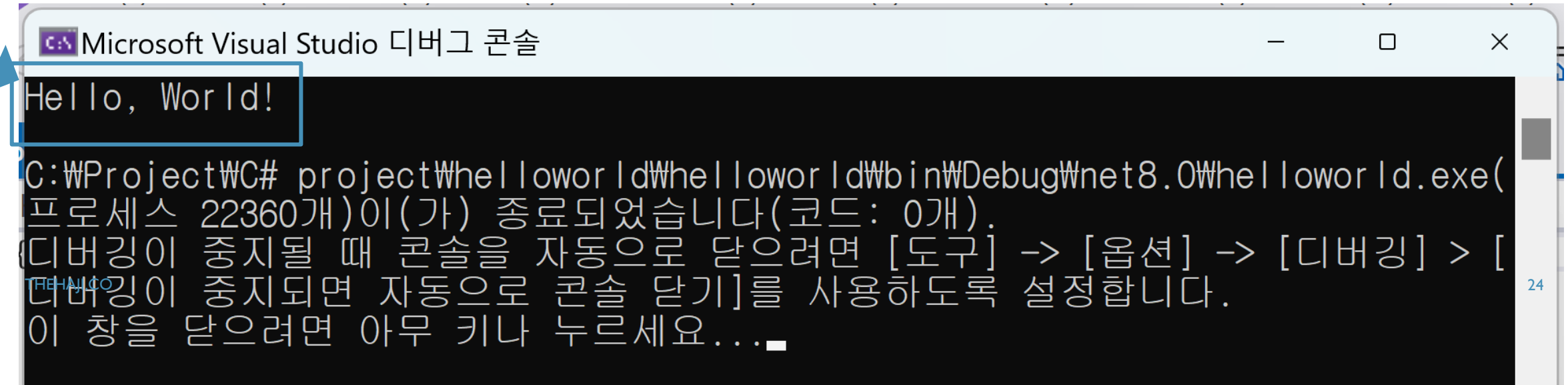
# 새 프로젝트 생성



```
Program.cs  + X
C# helloworld
1 // See https://aka.ms/new-console-template
2   Console.WriteLine("Hello, World!");
3
```

## ① 참고

.NET 6부터는 콘솔 템플릿을 사용하는 새 프로젝트가 이전 버전과는 다른 코드를 생성합니다. 자세한 내용은 새 C# 템플릿 생성 최상위 문 페이지를 참조하세요.



```
Microsoft Visual Studio 디버그 콘솔
Hello, World!
C:\WP\project\helloworld\helloworld\bin\Debug\net8.0\helloworld.exe(
프로세스 22360개)이(가) 종료되었습니다(코드: 0개).
{디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [
디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요....
```



# 새 프로젝트 생성

## ① 참고

.NET 6부터는 콘솔 템플릿을 사용하는 새 프로젝트가 이전 버전과는 다른 코드를 생성합니다. 자세한 내용은 새 C# 템플릿 생성 최상위 문 페이지를 참조하세요.

.NET 6부터 새 C# 콘솔 응용 프로젝트 템플릿은 *Program.cs* 파일에 다음 코드를 생성합니다.

```
C# 복사

using System;

namespace MyApp // Note: actual namespace depends on the project name.
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
THEHAJI.CO
```

```
C# 복사

// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

변수 타임

## 자료형

자료형	상세	표현 범위
int	부호있는 4byte 범위 정수	-2147483648 ~ 2147483647 (-21억 ~ 21억)
char	2byte 유니코드 문자	0000 ~ FFFF
float	4byte 부동 소수점	
double	8byte 부동 소수점	
bool	1byte	true, false
string		문자 여러 개 표현가능

# 전체 자료형

- 정수 자료형

자료형	크기	범위
sbyte	부호있는 1byte 범위 정수	-128 ~ 127
byte	부호없는 1byte 범위 정수	0 ~ 255
short	부호있는 2byte 범위 정수	-32768 ~ 32767
ushort	부호없는 2byte 범위 정수	0 ~ 65535
int	부호있는 4byte 범위 정수	-2147483648 ~ 2147483647 (-21억 ~ 21억)
uint	부호없는 4byte 범위 정수	0 ~ 4294967295 (~ 42억)
long	부호있는 8byte 범위 정수	-9223372036854775808 ~ ~9223372036854775807
ulong	부호없는 8byte 범위 정수	0~18446744073709551615
char	2byte 유니코드 문자	0000~FFFF

- 실수 자료형

자료형	크기	범위
float	4byte 부동 소수점 수	$\pm 1.5 \times 10^{-45} \sim \pm 3.4 \times 10^{38}$
double	8byte 부동 소수점 수 (float형 보다 정밀)	$\pm 5.0 \times 10^{-324} \sim \pm 1.7 \times 10^{308}$
decimal	16byte 부동 소수점 수 (double형 보다 정밀)	$1.0 \times 10^{-28} \sim \text{약 } 7.9 \times 10^{28}$

- 논리형

자료형	크기	범위
bool	1byte	true, false

- 문자형

자료형	크기	범위	표현방법
char	유니코드 2byte	0000 ~ FFFF	' ' (문자 1개만 표현 가능)
string			" " (문자 여러 개 표현 가능)

## 변수 변환

### 명시적

```
sbyte value1 = 64;  
sbyte value2 = 64;  
int value3 = (int)value1 + value2;  
Console.WriteLine(value3);
```

### 암시적

```
sbyte value1 = 64;  
sbyte value2 = 64;  
int value3 = value1 + value2;  
Console.WriteLine(value3);
```

## 연산자

연산자	사용법	
증가 연산자	num++	
감소 연산자	num--	
논리곱(&&, AND)	A && B	두조건이 전부 참일경우, 참
논리합(  , OR)	A    B	하나라도 참일 경우, 참
논리 부정(!, NOT)	! C	True <=> False
비트 연산	<< >>	

## 조건, 반복문

## 조건 반복문

명령어	
IF	만약 조건문이 해당한다면
switch	변수가 해당하는 것에
삼항 연산자 (조건? A, B)	조건문에 해당, 해당하지 않을 경우
for	조건 만큼 반복
while	조건에 해당할때까지 반복
foreach	순서에 상관없이 for
break	중간에 종료
continue	중간에서 다음 반복문으로 넘김



# IF

```
int num = 0;
```

```
if (num > 0)
```

```
    Console.WriteLine("양수");
```

```
else if (num < 0)
```

```
    Console.WriteLine("음수");
```

```
else
```

```
    Console.WriteLine("영");
```

# SWITCH

```
int input = 11;

switch (input)
{
    case 12:
        Console.WriteLine("input의 값이 12입니다.");
        break;
    default:
        Console.WriteLine("해당하는 값이 없습니다.");
        break;
}
```

THEHAJI.CO

```
string day = "화요일";
```

```
switch (day)
{
    case "월요일":
    case "화요일":
    case "수요일":
    case "목요일":
    case "금요일":
        Console.WriteLine("평일입니다.");
        break;
    case "토요일":
    case "일요일":
        Console.WriteLine("휴일입니다.");
        break;
}
```

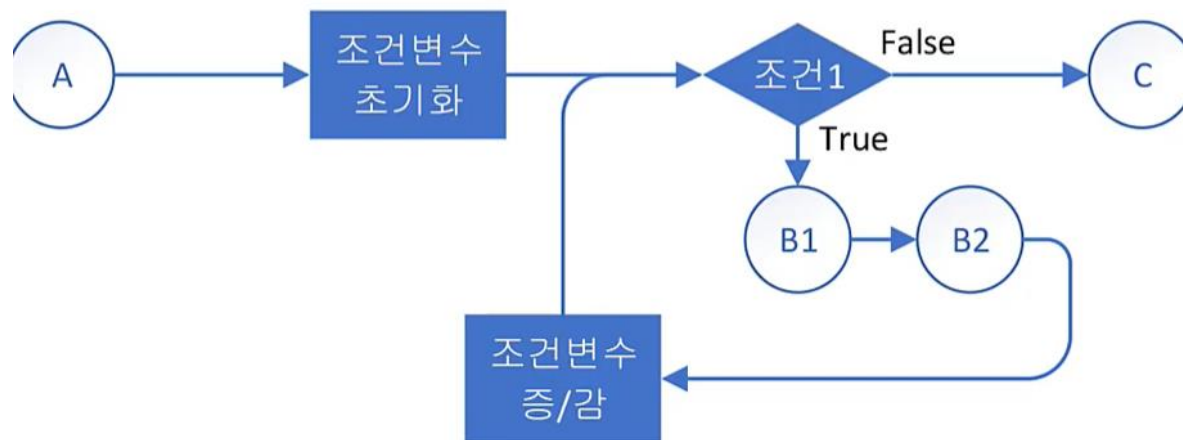
## 삼항 연산자

```
int number = 2;  
bool isEven;  
  
isEven = (number % 2 == 0) ? true : false;  
Console.WriteLine(isEven);
```

# FOR

```
for (①초기화; ②조건식; ④반복식)
{
    // ③반복 코드
}
```

```
for(int i=1;i<11;i++)
    Console.WriteLine(i);
```



# WHILE

```
int i=1;
```

```
while(i<11)
```

```
    Console.WriteLine(i++);
```

```
    Console.WriteLine("A");
```

```
    do
```

```
    {
```

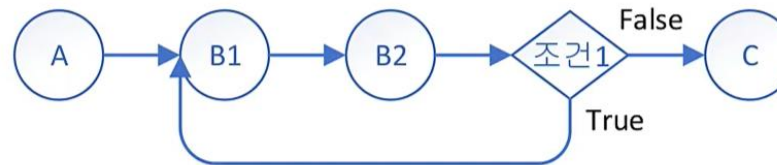
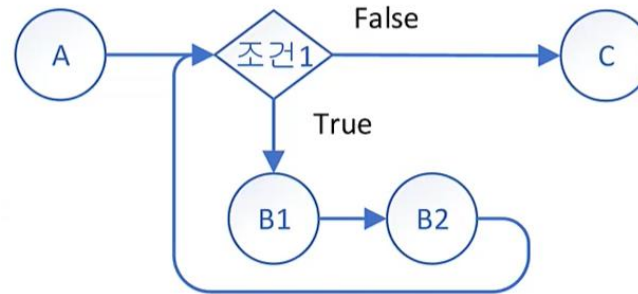
```
        Console.WriteLine(i++);
```

```
        Console.WriteLine("B");
```

```
    }
```

```
    while (i<10)
```

```
        Console.WriteLine("C");
```



# FOREACH

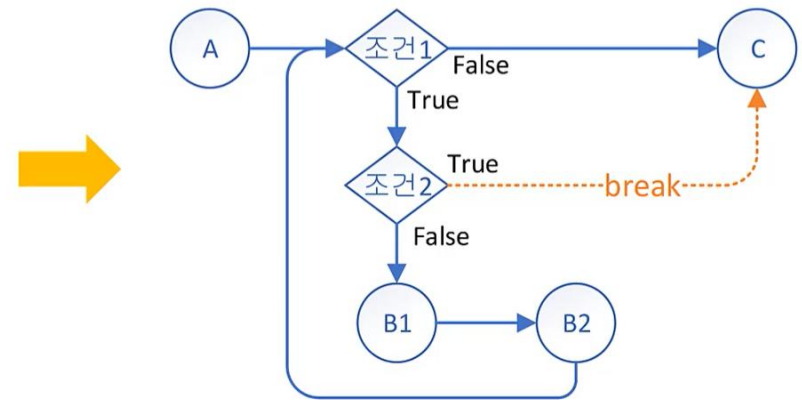
```
string text = "abcde";  
foreach (char a in text)  
{  
    Console.WriteLine(a);  
}
```

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
foreach (int i in numbers)  
{  
    Console.WriteLine(numbers[i] + " ");  
}
```

# BREAK

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 5)  
    {  
        break;  
    }  
    Console.WriteLine(i);  
}
```

```
Console.WriteLine("A");  
while (조건1)  
{  
    if (조건2)  
        break;  
    Console.WriteLine("B1");  
    Console.WriteLine("B2");  
}  
Console.WriteLine("C");
```



# CONTINUE

```
for (int i = 0; i < 10; i++)
```

```
{
```

```
    if (i % 2 == 0)
```

```
    {
```

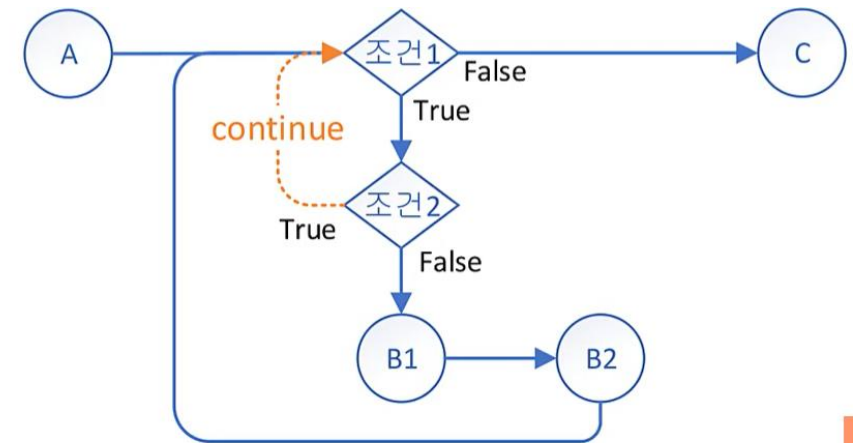
```
        continue;
```

```
    }
```

```
    Console.WriteLine (i+" : 홀수");
```

```
}
```

```
Console.WriteLine("A");  
while (조건1)  
{  
    if (조건2)  
        continue;  
  
    Console.WriteLine("B1");  
    Console.WriteLine("B2");  
}  
Console.WriteLine("C");
```





# GOTO

```
Console.WriteLine ("A");
```

```
if (true)
```

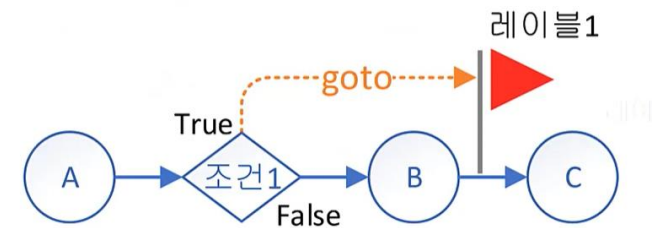
```
    goto label I;
```

```
    Console.WriteLine ("B");
```

```
label I:
```

```
    Console.WriteLine ("C");
```

```
Console.WriteLine ("A");  
if (조건1)  
    goto 레이블1;  
Console.WriteLine ("B");  
레이블1:  
    Console.WriteLine ("C");
```



## 추가 문법

# CONST 상수 선언

- `const int MAX_INT_BIT = 32;`

# ENUM 열거형식

기본적으로 열거형 멤버의 연결된 상수 값은 int 형식입니다. 즉, 0으로 시작하고 정의 텍스트 순서에 따라 1씩 증가합니다.

```
enum Season
{
0    Spring,
1    Summer,
2    Autumn,
3    Winter
}
```

## ENUM 열거형식 2

```
public enum Days
{
    None    = 0b_0000_0000, // 0
    Monday  = 0b_0000_0001, // 1
    Tuesday = 0b_0000_0010, // 2
    Wednesday = 0b_0000_0100, // 4
    Thursday = 0b_0000_1000, // 8
    Friday   = 0b_0001_0000, // 16
    Saturday = 0b_0010_0000, // 32
    Sunday   = 0b_0100_0000, // 64
    Weekend  = Saturday | Sunday
}
```

```
Days meetingDays = Days.Monday | Days.Wednesday | Days.Friday;
```

```
Console.WriteLine(meetingDays);
```

```
// Output:
```

```
// Monday, Wednesday, Friday
```

```
Days workingFromHomeDays = Days.Thursday | Days.Friday;
```

```
Console.WriteLine($"Join a meeting by phone on {meetingDays & workingFromHomeDays}");
```

```
// Output:
```

```
// Join a meeting by phone on Friday
```

```
bool isMeetingOnTuesday = (meetingDays & Days.Tuesday) == Days.Tuesday;
```

```
Console.WriteLine($"Is there a meeting on Tuesday: {isMeetingOnTuesday}");
```

```
// Output:
```

```
// Is there a meeting on Tuesday: False
```

```
var a = (Days)37;
```

```
Console.WriteLine(a);
```

```
// Output:
```

```
// Monday, Wednesday, Saturday
```

# 배열

변수 선언, 메모리 할당

```
int[] arr = new int[2];
```

값 초기화

```
int[] arr2 = new int[2] {1,2};
```

배열의 길이 만큼 for문

```
int[] arr = {1,2,3,4,5};  
for(int i =0; i<arr.Length;i++)  
{  
    print(i);  
}  
Console.WriteLine(Array.IndexOf(arr, 3));
```

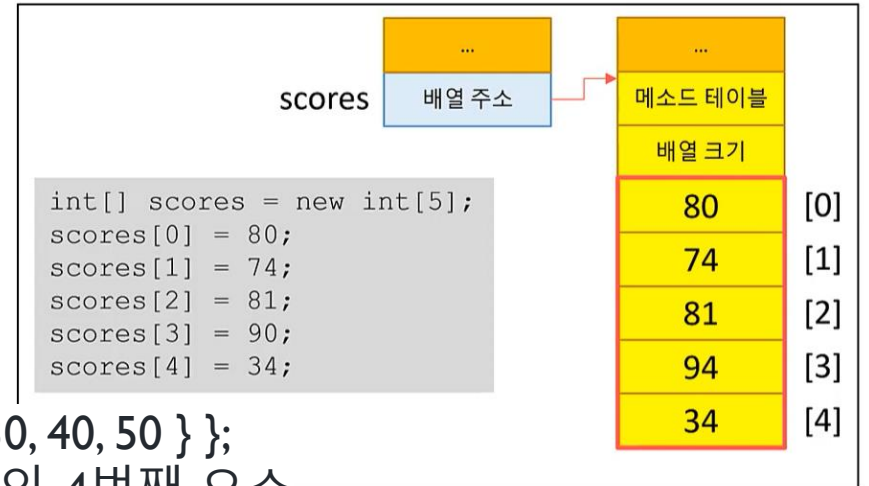
THEHAJI.CO

## 다차원 배열

```
int[,] array2 = { { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } };  
print(array2[1, 3]); // 2번째 1차원 배열의, 4번째 요소  
print(array2[0, 1]); // 1번째 1차원 배열의, 2번째 요소
```

## 삼차원 배열

```
int[,,] array3 = {  
    { { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } },  
    { { 1, 2, 3, 4, 5 }, { 10, 20, 30, 40, 50 } }  
};
```



## 컬렉션 - ARRAYLIST 클래스

```
ArrayList al = new ArrayList();  
    // Add 메소드를 통해 ArrayList에 아이템 추가  
    al.Add(1);  
    al.Add("Hello");  
    al.Add(3.3);  
    al.Add(true);  
    foreach(var item in al) {  
        Console.WriteLine(item);  
    }  
    Console.WriteLine();  
    // Remove 메소드를 통해 ArrayList에서 아이템  
    삭제  
    al.Remove("Hello");  
    foreach(var item in al) {  
        Console.WriteLine(item);  
    }
```

# 컬렉션 – QUEUE 클래스

```
Queue qu = new Queue();
```

```
// Enqueue 메소드를 통해 Queue에 아이템 추가
```

```
qu.Enqueue(1);
```

```
qu.Enqueue(2);
```

```
qu.Enqueue(3);
```

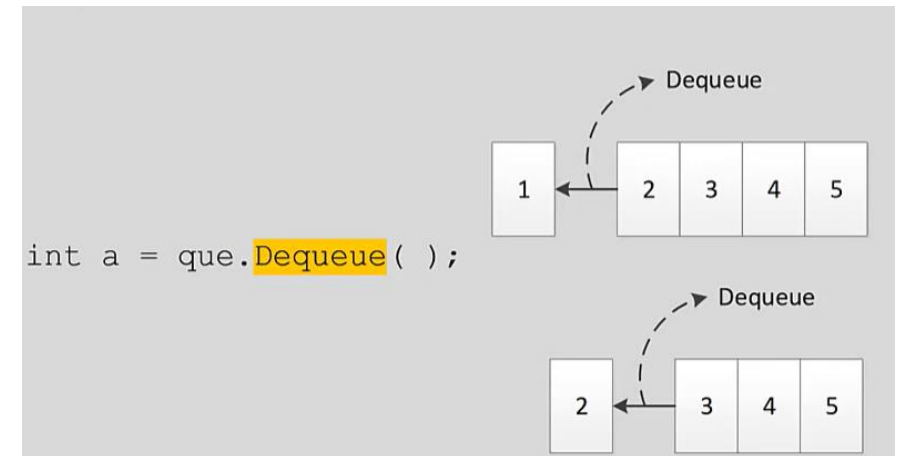
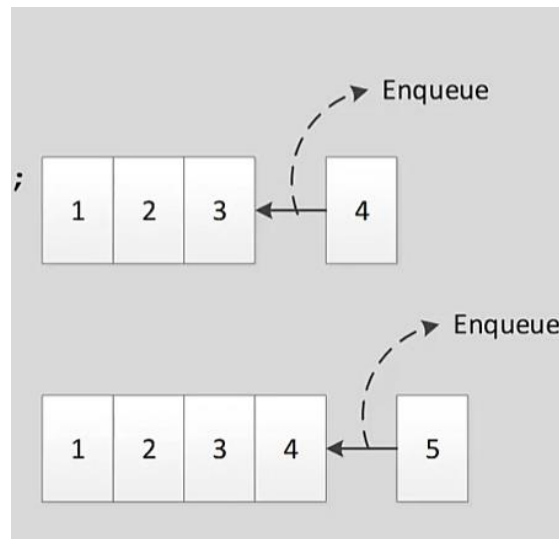
```
// Dequeue 메소드를 통해 Queue에서 아이템을 제거
```

```
while (qu.Count > 0)
```

```
{
```

```
    Console.WriteLine(qu.Dequeue());
```

```
}
```





## 컬렉션 - STACK 클래스

```
Stack st = new Stack();
```

```
// Push 메소드를 통해 Stack에 아이템 추가
```

```
st.Push(1);
```

```
st.Push(2);
```

```
st.Push(3);
```

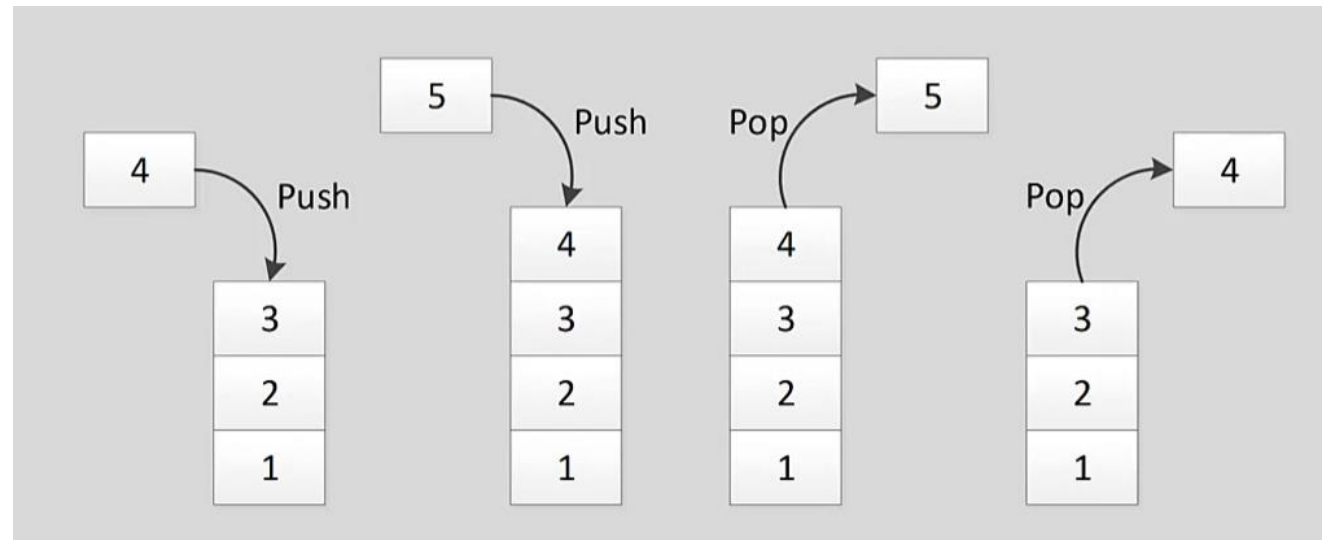
```
// Pop 메소드를 통해 Stack에서 아이템을 제거
```

```
while (st.Count > 0)
```

```
{
```

```
    Console.WriteLine(st.Pop());
```

```
}
```



## 컬렉션 - HASHTABLE 클래스

Hashtable 클래스는 키(Key)와 값(Value)으로 저장하는 자료 구조

```
Hashtable ht = new Hashtable();
```

```
// 키와 값 형태로 Hashtable에 아이템 추가
```

```
ht["apple"] = "사과";
```

```
ht["banana"] = "바나나";
```

```
ht["orange"] = "오렌지";
```

```
// Hashtable에 저장된 키에 해당하는 값을 출력
```

```
Console.WriteLine(ht["apple"]);
```

```
Console.WriteLine(ht["banana"]);
```

```
Console.WriteLine(ht["orange"]);
```

## 실습 1. 계산기 만들기

- 사칙 연산이 가능한 console 계산기

```
x = Convert.ToInt32(Console.ReadLine());
```

```
y = Convert.ToInt32(Console.ReadLine());
```

```
result = add(x, y);
```

## 예외 처리 : TRY ~ CATCH 문

```
Console.Write("나눌 숫자를 입력하세요 : ");  
int divider = int.Parse(Console.ReadLine());  
Console.WriteLine(10 / divider);
```

예외가 처리되지 않음

**System.DivideByZeroException:** 'Attempted to divide by zero.'

[호출 스택 표시](#) | [자세히 보기](#) | [세부 정보 복사](#) | [Live Share 세션을 시작합니다](#)

▶ 예외 설정

## 예외 처리 : TRY ~ CATCH 문

```
Console.WriteLine("10을 0으로 나눕니다. ");  
int divider = 0;  
try  
{  
    Console.WriteLine(10 / divider);  
    Console.WriteLine("0으로 나눌 수 있습니다.");  
}  
catch  
{  
    Console.WriteLine("0으로 나눌 수 없습니다.");  
}
```

```
int divider = 0;  
  
try  
{  
    Console.WriteLine(10 / divider);  
    Console.WriteLine("0으로 나눌 수 있습니다.");  
}  
catch (Exception e)  
{  
    Console.WriteLine("예외 상황 : " + e.Message);  
    // 예외 상황 : Attempted to divide by zero.  
}
```

## 객체 지향 언어

클래스명

접근 제한자

클래스

```
using UnityEngine;
namespace Coderzero.Example
{
    public class ClassExample : MonoBehaviour
    {
        public string m_PublicField = string.Empty;
        private int m_PrivateField = 0;

        public int m_Property { get; set; }

        void Start()
        {
            PrivateMethod();
        }

        void Update()
        {
            PublicMethod();
        }

        public void PublicMethod()
        {

        }

        private void PrivateMethod()
        {

        }
    }
}
```

Using 지시문

네임스페이스

상속

필드, 멤버변수

필드, 멤버변수

필드, 속성

MonoBehaviour  
이벤트 함수

필드,  
Public 메소드

Private 메소드

# 클래스와 인스턴스

- Factory 에서 모형 틀 = 클래스
- 모형틀에 찍힌 빵 = 인스턴스

클래스

인스턴스

```
Person p1;  
p1 = new Person();  
// 인스턴스 변수를 통해 속성과 메서드 호출  
p1.Name = "서준";  
p1.Eat();  
class Person  
{  
    // 속성 변수의 값을 null로 초기화  
    public string Name = null;  
    public string Birthday = null;  
    public string Gender = null;  
  
    // 메서드 구현  
    public void Eat()  
    {  
        Console.WriteLine(Name + "이(가) 아침을 먹습니다.");  
    }  
    public void Walk()  
    {  
        Console.WriteLine(Name + "이(가) 걷습니다.");  
    }  
    public void Run()  
    {  
        Console.WriteLine(Name + "이(가) 뛸니다.");  
    }  
}
```



## 클래스 소멸 주기



생성자(Constructor)

인스턴스 생성시 호출  
생성자가 오버로딩 되어있어  
다중 생성자가 있을 경우,  
호출한 생성자만 실행.

소멸자(Destructor)  
== 종료자(Finalizer)

소멸자도 호출되는 시점을 개발자가  
결정하는 게 아닌 시스템의 가비지  
컬렉터(Garbage Collector)라는  
소프트웨어가 결정

## 생성자와 소멸자(종료자)

```
static void Test()
{
    Cat myCatA = new Cat();
    Cat myCatB = new Cat("하루");
    Cat myCatC = new Cat("코코");
    Cat myCatD = new Cat("몰리", 3);

    // Cat myCatE = new Cat("몰리", 3.5F);
}

Test();
GC.Collect();
Console.ReadLine();
```

```
class Cat
{
    public string Name;
    public int Age;

    // 생성자
    public Cat()
    {
        Name = "길고양이";
        Console.WriteLine("길고양이 생성자가 호출되었습니다.");
    }

    public Cat(string name)
    {
        Name = name;
        Console.WriteLine("고양이의 이름은 " + Name + "입니다.");
    }

    public Cat(string name, int age)
    {
        Name = name;
        Age = age;
        Console.WriteLine("고양이의 이름은 " + Name + "이며, 나이는 " + Age +
            "kg입니다.");
    }

    // 소멸자(종료자)
    ~Cat()
    {
        Console.WriteLine(Name + "가 사라집니다.");
    }
}
```

## 소멸자(DESTRUCTOR) == 종료자(FINALIZER)

소멸자는 .NET 5(.NET Core 포함) 이상 버전인 경우 애플리케이션이 종료될 때 소멸자를 호출하지 않습니다. 그러므로 콘솔 창에서 소멸자가 호출되었는지 확인할 수 없습니다. .NET 5(.NET Core 포함) 이상 버전에서 소멸자를 강제로 실행하기 위해서는 GC.Collect()를 호출해야 합니다.

# 인터페이스와 추상 클래스

C# 인터페이스와 추상클래스의 차이점

	Interface	Abstract Class
접근 지정자	<ul style="list-style-type: none"><li>- 함수에 대한 접근 지정자를 가질수 없습니다.</li><li>- 기본적으로 public 입니다.</li></ul>	<ul style="list-style-type: none"><li>- 함수에 대한 접근 지정자를 가질 수 있습니다.</li></ul>
구현	<ul style="list-style-type: none"><li>- 구현이 아닌 서명만 가질 수 있습니다.</li></ul>	<ul style="list-style-type: none"><li>- 구현을 제공할 수 있습니다.</li></ul>
속도	<ul style="list-style-type: none"><li>- 인터페이스가 상대적으로 느립니다.</li></ul>	<ul style="list-style-type: none"><li>- 추상 클래스가 빠릅니다.</li></ul>
인스턴스화	<ul style="list-style-type: none"><li>- 인터페이스는 추상적이며 인스턴스화 할 수 없습니다.</li></ul>	<ul style="list-style-type: none"><li>- 추상클래스는 인스턴스화 할 수 없습니다.</li></ul>
필드	<ul style="list-style-type: none"><li>- 인터페이스는 필드를 가질 수 없습니다.</li></ul>	<ul style="list-style-type: none"><li>- 추상클래스는 필드와 상수를 정의 할 수 있습니다.</li></ul>
메소드	<ul style="list-style-type: none"><li>- 인터페이스에는 추상메소드만 있습니다.</li></ul>	<ul style="list-style-type: none"><li>- 추상클래스에는 비추상메소드가 있을 수 있습니다.</li></ul>

# 인터페이스와 상속

