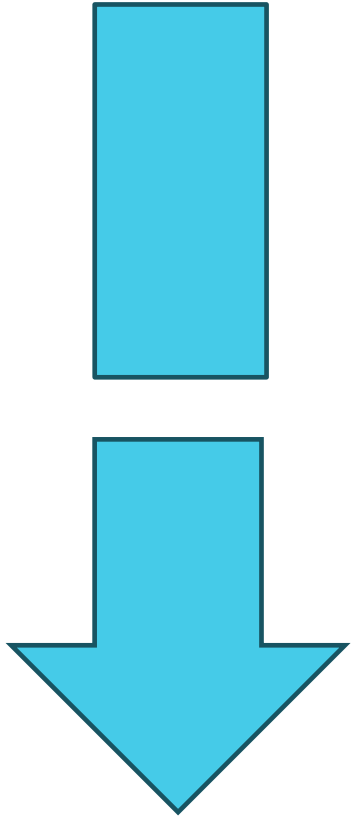
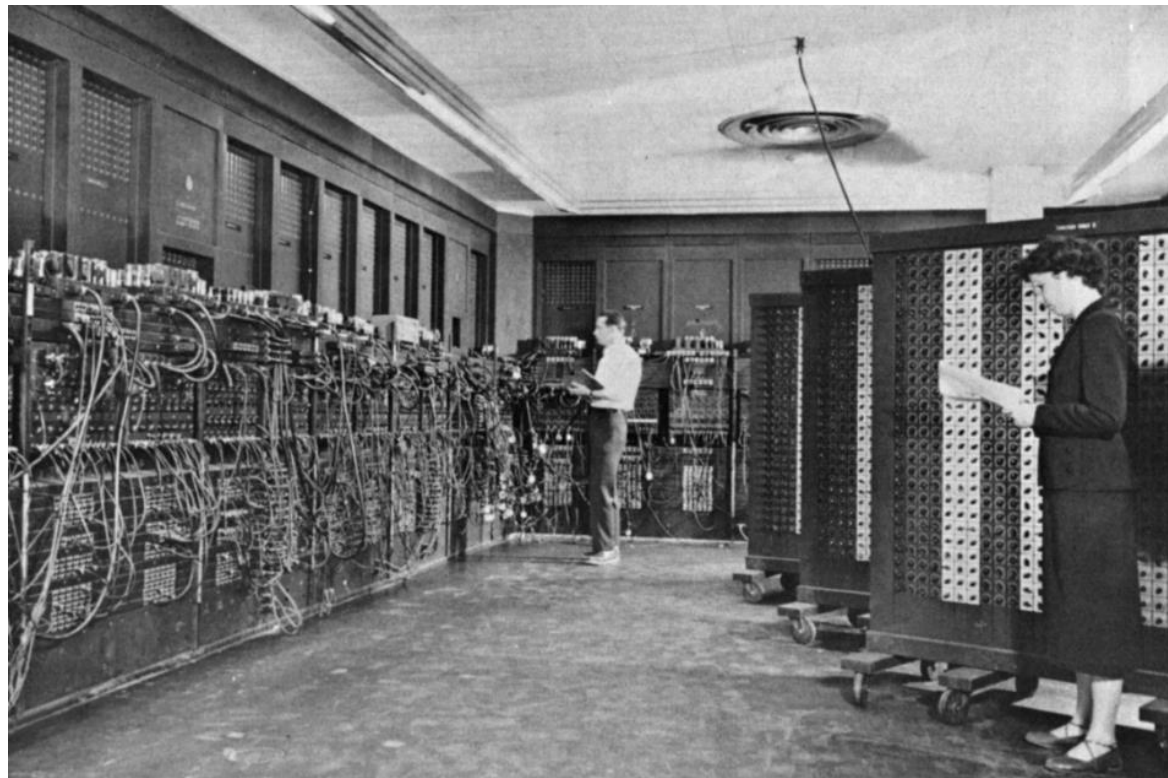


C# 언어의 발전 과정

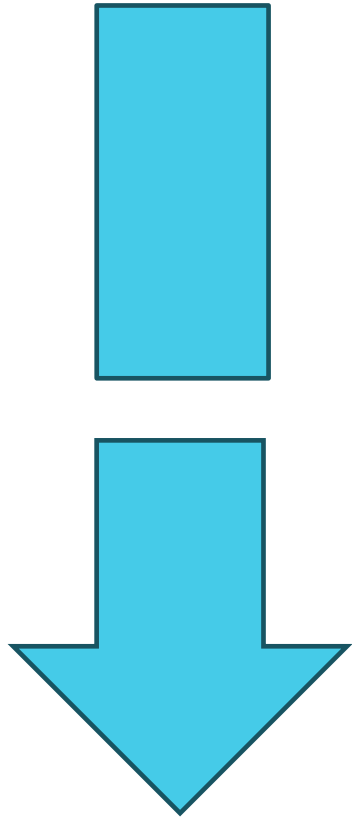
C#의 역사



THEHAJI.CO



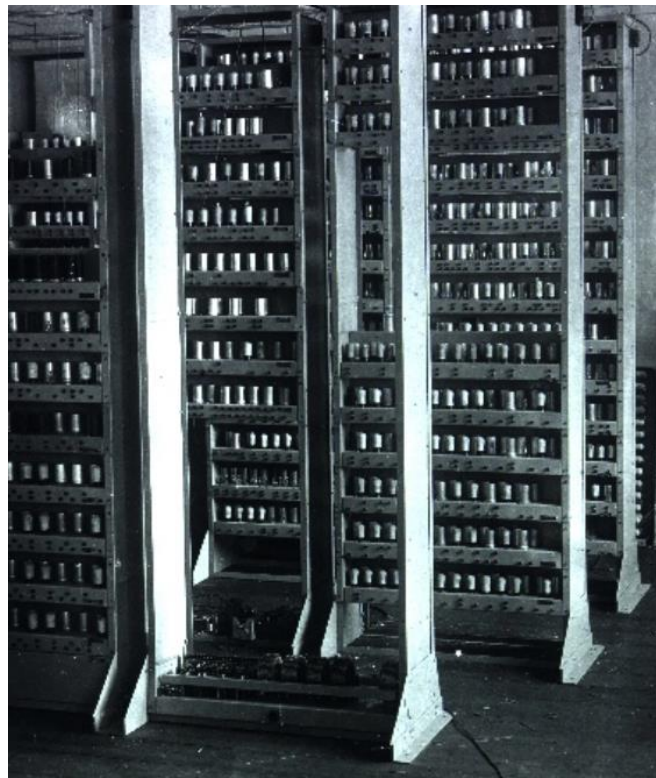
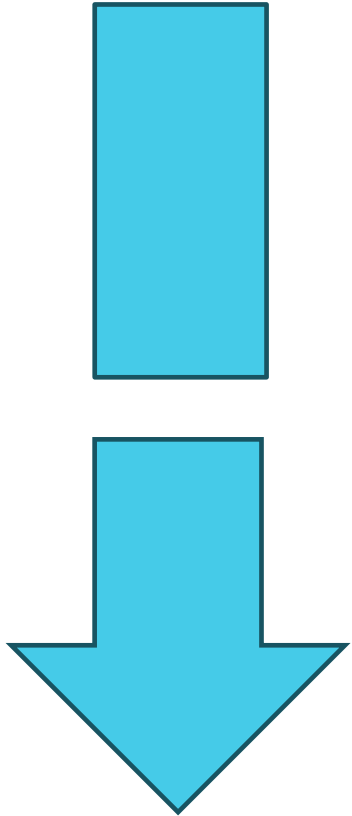
1946년 애니악(ENIAC)



폰 노이만

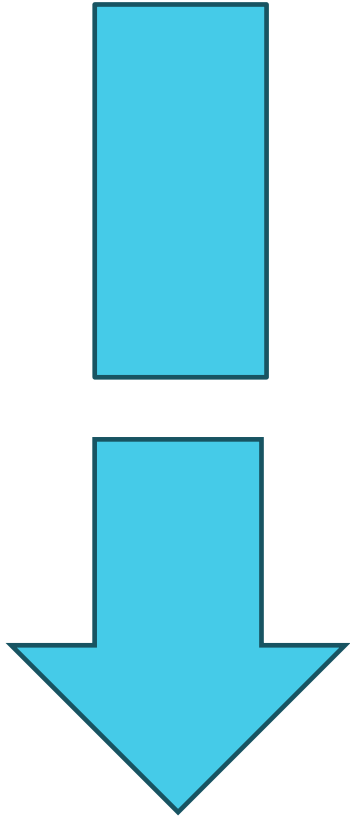
- <https://youtu.be/Yaqv27AqXml?si=7VLM-KXsREMYqjUz&t=469>

C#의 역사

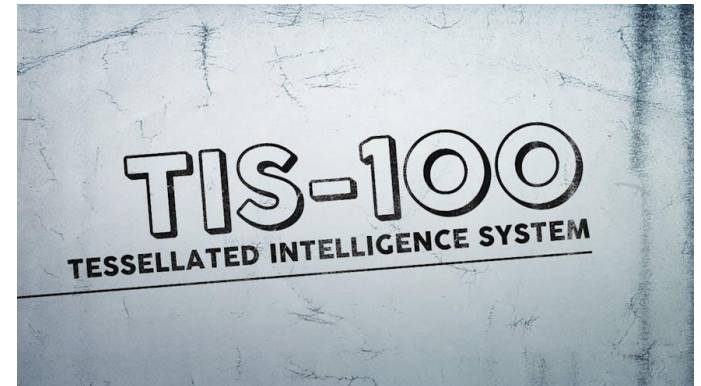


최초의 프로그램 내장식 컴퓨터 EDSAC에드삭(폰 노이만 설계)

C#의 역사

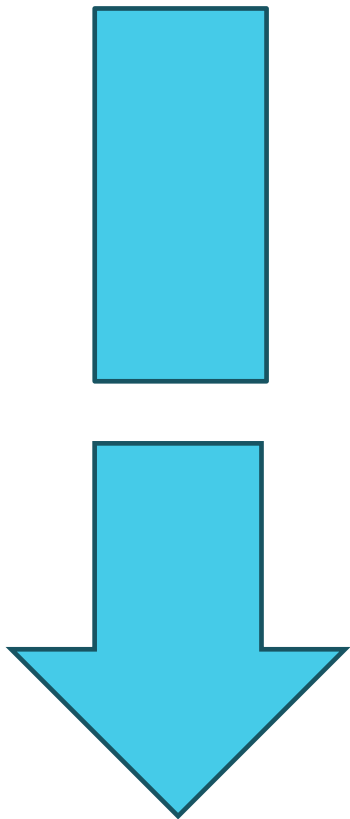


- 어셈블리어 등장
- 진공관에서 트랜지스터로 발전
 - 1928년 포트란/ 컴파일러 등장
- 1964년 BASIC 언어의 등장



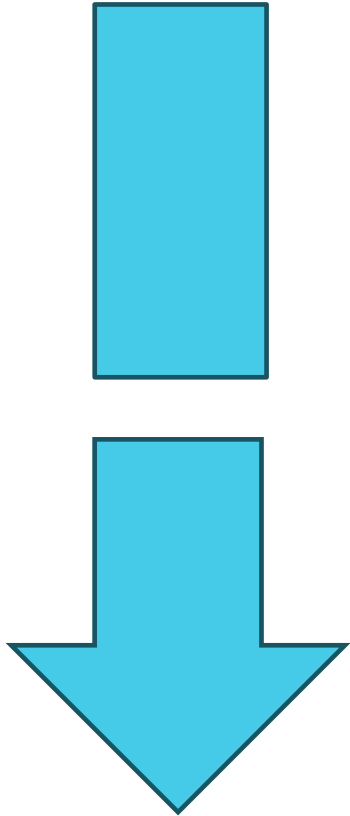
* 어셈블리어 게임으로 즐기기

C#의 역사



- 1972년 C언어 등장 (데니스 리치)
 - 절차 지향 프로그래밍
- 유닉스 개편 (C언어 기반으로)
- 1985년 C++ 언어 등장
 - C언어에서 객체 지향 프로그래밍으로 바뀜
- 1991년 JAVA 등장 (제임스 고슬링)
- 1995년 JAVASCRIPT 발표

C#의 역사



- 2002년 1월
 - C# 언어의 등장
 - C# 버전 1.0

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>

2002년 1월
C# 버전 1.0

Visual Studio.Net 2002와 함께 릴리스된 C# 버전 1.0은 Java와 매우 비슷했습니다.
ECMA에 대한 명시된 디자인 목표의 일환으로 "단순하고 현대적인 범용 개체 지향 언어"가 되려고 했습니다.
당시 Java처럼 보이는 것은 초기 디자인 목표를 달성했음을 의미했습니다.

C# 버전 1.0은 Windows 플랫폼에서 Java를 대체하는 실용적인 방법이었습니다.

- 클래스
- 구조체
- 인터페이스
- 이벤트
- 속성
- 대리자
- 연산자 및 식
- 문
- 특성

C#의 역사

- Delegate(대리자)

대리자의 선언과 사용

- **delegate** 키워드를 이용하여 선언
- 메소드와 같이 대리자 또한 매개변수 목록과 반환 형식을 가짐

한정자 **delegate** 반환형식 델리게이트이름 (매개변수_목록);

- 대리자의 선언과 사용 예

```
private delegate int MyDelegate( int a, int b );
```

```
MyDelegate Callback;
```

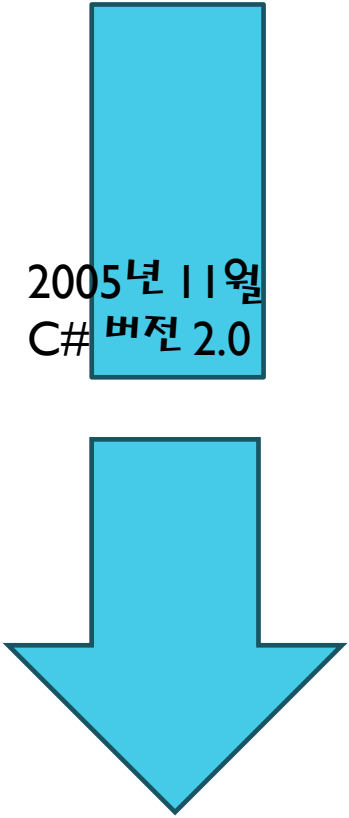
```
Callback = new MyDelegate( Plus );  
Console.WriteLine( Callback( 3, 4 ) ); // 7 출력
```

```
Callback = new MyDelegate( Minus );  
Console.WriteLine( Callback( 7, 5 ) ); // 2 출력
```

```
int Plus( int a, int b )  
{  
    return a + b;  
}  
  
int Minus( int a, int b )  
{  
    return a - b;  
}
```

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2005년 11월
C# 버전 2.0

■ 제네릭

예를 들어 `List<T>`를 사용하면 `List<string>` 또는 `List<int>`를 사용하고 이를 반복하는 동안 해당 문자열이나 정수에 형식이 안전한 작업을 수행할 수 있습니다.

C#은 Java를 따라잡으려는 노력을 계속했습니다. Java는 이미 제네릭 및 반복기가 포함된 버전을 출시했습니다. 하지만 언어가 계속 발전함에 따라 곧 변경될 것입니다.

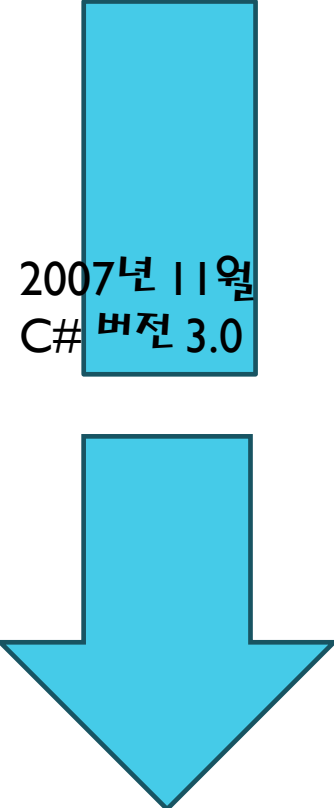
- 제네릭(일반화)
- 부분 형식(Partial Type)
- 무명 메서드
- Nullable 값 형식
- 반복기
- 공변성(Covariance) 및 반공변성(Contravariance)

C#의 역사

- 제네릭(일반화)
 - class **GenericClass** < **T** >
- Nullable ~~값~~ 형식
 - `int?` var1 = 10;
 - `int?` var2 = null;

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2007년 11월
C# 버전 3.0

- C# 3.0은 C#을 하이브리드 개체 지향/기능 언어로 전환하기 위한 토대를 마련
- C# 버전의 핵심 기능은 LINQ(Language-Integrated Query)라고도 하는 쿼리 식
- 컬렉션에 대한 작업을 수행할 수 있는 SQL 스타일의 선언적 쿼리를 작성할 수 있습니다
 - for 루프를 작성하는 대신 이제 간단하게 list.Average()로 처리

- 자동 구현 속성
- 무명 형식
- 쿼리 식
- 람다 식
- 식 트리
- 확장 메서드
- 암시적 형식 지역 변수
- 부분 메서드
- 개체 및 컬렉션 이니셜라이저

C#의 역사

■ 자동 구현 속성

- class **Human** {
 - public string name { get; set; }
 - public int age { get; set; }
- }

■ 람다 식

- 매개변수(입력) => 식(출력)

■ 암시적 형식 지역 변수

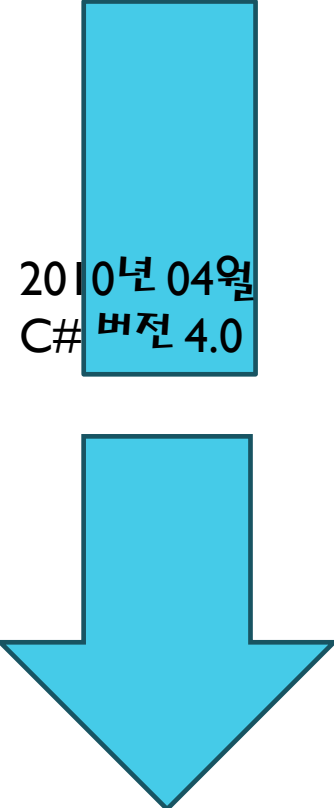
- var 키워드

람다식 지원

- * Java 8 (2014년 3월)
- * C++ 11 (2011년 8월)
- * C# 3.0 (2007년 11월)

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2010년 04월
C# 버전 4.0

dynamic 키워드였습니다. dynamic 키워드는 C# 버전 4.0에 컴파일 시간에 컴파일러를 재정의하는 기능을 도입했습니다. 동적 키워드를 사용하면 JavaScript와 같이 동적으로 형식화된 언어와 유사한 구조를 만들 수 있습니다. dynamic x = "a string"을 만든 다음, 6을 추가하여 다음에 수행해야 할 작업을 런타임에 맡길 수 있습니다.

동적 바인딩은 오류를 유발할 수 있지만 언어 내에서 훌륭한 기능도 제공합니다.

- 동적 바인딩
- 명명된/선택적 인수
- 제네릭 공변(covariant) 및 반공변(contravariant)
- 포함된 interop 형식

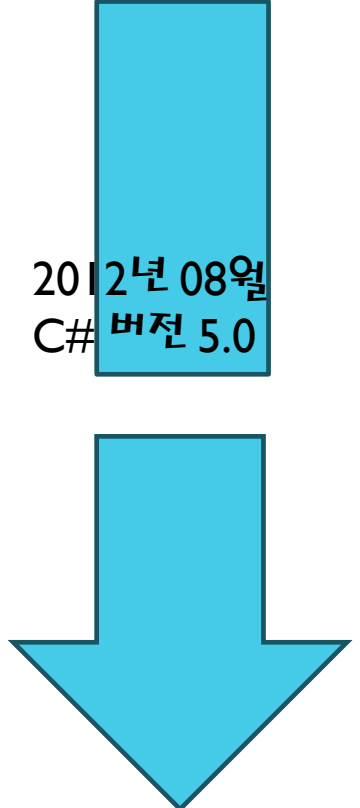
C#의 역사

- 동적 바인딩

- `dynamic variable = "Hello";`
- `Console.WriteLine(variable);`
- `variable = 123;`
- `Console.WriteLine(variable);`

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2012년 08월
C# 버전 5.0

- 비동기 프로그래밍을 위한 async 및 await 모델

- 비동기 멤버
- 호출자 정보 특성
- 코드 프로젝트: C# 5.0에서 호출자 정보 특성

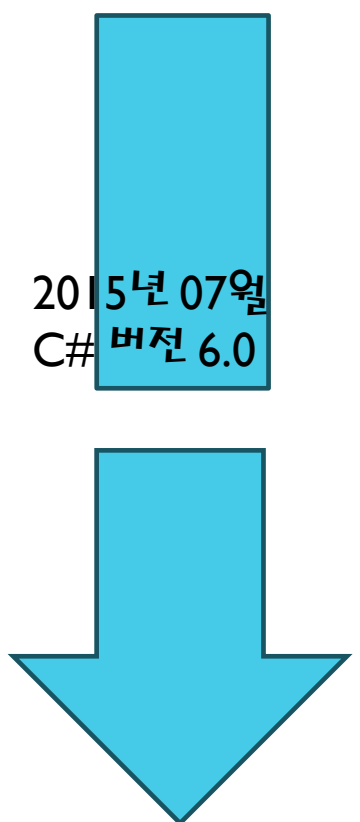
C#의 역사

- 비동기 멤버
- **async 및 await를 사용한 비동기 프로그래밍**
- **async Task**
- Egg eggs = **await** FryEggsAsync(2);
- Console.WriteLine("eggs are ready");



C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2015년 07월
C# 버전 6.0

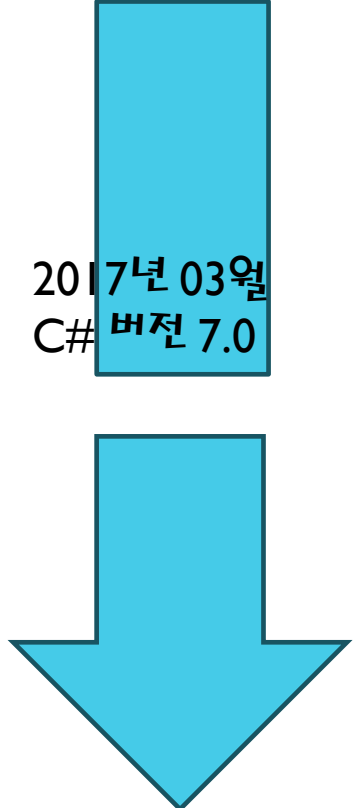
- 정적 가져오기
- 예외 필터
- Auto 속성 이니셜라이저
- 식 본문 멤버
- Null 전파자
- 문자열 보간
- nameof 연산자
- Catch/Finally 블록의 Await
- Getter 전용 속성의 기본값

C#의 역사

- 읽기 전용 자동 속성 선언하기 (Auto 속성 이니셜라이저)
 - `public string name { get; } = "Default Name";`
- C# 6.0 부터 \$ 기호를 사용해 문자열을 표현하는 방법이 추가 (문자열 보간)
- `Console.WriteLine($"{num1} + {num2} = {num1+num2}");`

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2017년 03월
C# 버전 7.0

- .NET Core는 이제 모든 운영 체제를 대상으로 하며 클라우드와 휴대성에 확실히 집중하고 있습니다.

- 외부 변수
- 튜플 및 분해
- 패턴 일치
- 로컬 함수
- 확장된 식 본문 멤버
- Ref locals
- Ref가 반환됩니다.
- Throw 식

- 튜플(Tuples)은 데이터들 그룹으로 묶어 제공 (튜플 및 분해)

-

```
(double, int) t1 = (4.5, 3);
```

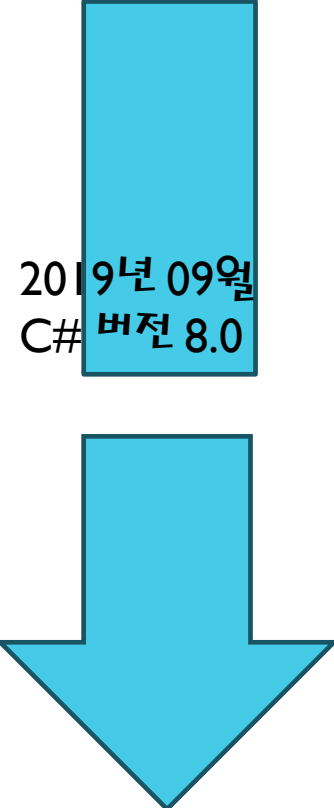
- Console.WriteLine(\$"튜플 내 항목은 {t1.Item1}과 {t1.Item2}입니다.");

- (double Sum, int Count) t2 = (4.5, 3);

- Console.WriteLine(\$"Count: {t2.Count}, Sum: {t2.Sum}.");

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2019년 09월
C# 버전 8.0

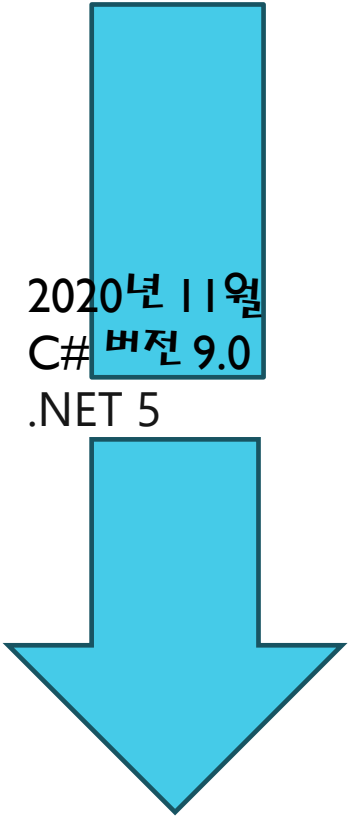
- 읽기 전용 멤버
- 기본 인터페이스 메서드
- 패턴 일치 향상된 기능:
 - Switch 식
 - 속성 패턴
 - 튜플 패턴
 - 위치 패턴
- using 선언
- 정적 로컬 함수
- 삭제 가능한 ref struct
- nullable 참조 형식
- 비동기 스트림
- 인덱스 및 범위
- null 병합 할당
- 관리되지 않는 생성 형식
- 중첩 식의 stackalloc
- 보간된 약어 문자열의 향상된 기능

- 해당 기능은 .NET Core 3.0용 CLR에 추가되었습니다.

- C# 8.0 부터 인덱스를 배열의 뒤에서 부터 찾을 수 있는 \wedge 연산자와 하나의 값이 아닌 그룹으로 값을 가져오기 위한 \dots 연산자를 지원 합니다.
 - (인덱스 및 범위)

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2020년 11월
C# 버전 9.0
.NET 5

- 레코드
- setter만 초기화 - INIT
- 최상위 문
- 패턴 일치 향상된 기능: 관계형 패턴 및 논리 패턴
- 성능 및 interop
 - 네이티브 크기의 정수
 - 함수 포인터
 - localsinit 플래그 내보내기 표시 안 함
 - 모듈 이니셜라이저
 - 부분 메서드에 대한 새로운 기능
- 기능 맞춤 및 완료
 - 대상으로 형식화된 new 식
 - static 익명 함수
 - 대상 형식 조건식
 - 공변 반환 형식
 - foreach 루프에 대한 확장 GetEnumerator 지원
 - 람다 dis카드 매개 변수
 - 로컬 함수의 특성

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



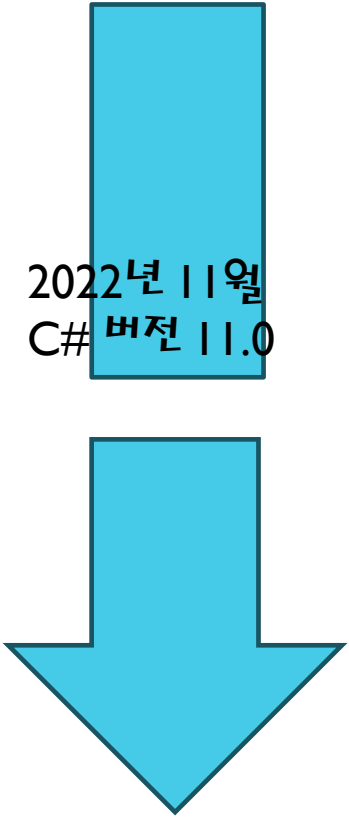
2021년 11월
C# 버전 10.0

- 레코드 구조체
- 구조체 형식 개선
- 보간된 문자열 처리기
- global using 지시문
- 파일 범위 네임스페이스 선언
- 확장 속성 패턴
- 람다 식 개선
- const 보간된 문자열 허용
- 레코드 형식은 ToString()을 봉인할 수 있음
- 한정된 할당 개선
- 동일한 분해에서 할당과 선언을 모두 허용
- 메서드의 AsyncMethodBuilder 특성을 허용
- CallerArgumentExpression 특성
- 향상된 #line pragma

- global using 지시문
- C# 10.0 부터는 global 키워드를 사용해 전체 프로젝트에서 필요한 네임 스페이스를 한 곳에서 관리할 수 있게 됩니다.

C#의 역사

- <https://learn.microsoft.com/ko-kr/dotnet/csharp/whats-new/csharp-version-history>



2022년 11월
C# 버전 11.0

- 원시 문자열 리터럴
- 일반 수학 지원
- 제네릭 특성
- UTF-8 문자열 리터럴
- 문자열 보간 식의 줄
- 목록 패턴
- 파일 로컬 형식
- 필수 멤버
- 자동 기본 구조체
- 상수의 패턴 일치 `Span<char>string`
- 확장된 nameof 범위
- Numeric IntPtr
- ref 필드 및 scoped ref
- 대리자로의 메서드 그룹 변환 개선
- 경고 웨이브 7