컴파일 언어와 인터프리터 언어(=동적 언어)의 차이

컴파일 언어

[종류]

```
C, C++, GO
```

[순서]

```
코딩(문서) -> 빌드 (컴파일러) -> 실행파일 (기계어)
```

[특징]

```
- 일방향
- 어셈블러는 역방향 가능
- 컴퍼일러 언어는 역방향이 불가능
- 이런 기계어가 나오는 언어가 컴파일 언어
- 실행하기 전에 기계어로 모두 변환 해놓고 실행됨
```

[장점]

```
- 속도가 빠름
-
- 어셈블리어로 미리 변환이 되어있어서 바로 실행하기만 하면 됨
- 아직까지 프로그램의 핵심, 근본이 되는 부분의 경우 속도가 매우 중요하여
- C와 C++이 많이 사용됨
```

[단점]

```
- 플랫폼 별 다른 변환이 필요함

>>OS

- 각 OS 마다 동작 방식이 다름

- 실행 파일을 생성하는 방식도 다르고 실행하는 방식도 다름

>>CPU 칩셋

- CPU 종류마다 어셈블리언어가 다름

- CPU 종류가 같더라도 CPU 가 발전하면서 새로운 op code 가 추가됨

- 그러므로 칩셋에 따라 어떻게 변환하느냐에 따라 효율이 달라짐

- 그래서 같은 CPU 계열 이라도 변화방법이 달라지기도 함
```

동적 언어(인터프리터언어)

[종류]

```
C#, JAVA, Python, R
```

[순서]

```
코딩(문서) -> 빌드 (JAVA, C#) -> 중간 언어 -> 기계어
```

[특징]

```
- 실행하는 중간에 변환을 함

- 기계어로 변환하는 과정없이 한줄 한줄 해석하여 바로 명령어를 실행
```

[장점]

- 생산성이 좋다
- 플랫폼에 상관없이 사용가능
- 프로그램을 만들어 놓으면 실행할 때 실행하는 주체가 어떤 플랫폼인지 파악해서
- 그것에 맞게 그때그때 변환해서 실행하는 방식이므로
- 플랫폼에 상관없이 동작함

예)

*JAVA : class 파일만 있으면 JVM 위에서 동작하기 때문에 OS 영향을 받지 않음

*C# : 기존에는 Windows 에서만 동작했지만 결국 Open 하여 다양한 플랫폼에서 사용가능

[단점]

- 속도가 느림
- 실행하는 중간중간 기계어로 변환을 해야해서
- 상대적으로 컴파일 언어보다 느림
- 하지만 하드웨어의 발전으로 컴파일 언어와 동적 언어와 속도차이가 많이 좁혀진 상태임