

01

프로그래밍 언어의 개요

01. 프로그래밍 언어의 개요

I. 프로그래밍 언어의 개념

- 프로그램 : 컴퓨터를 이용해 문제를 해결할 때 컴퓨터에 내리는 명령어들의 집합.
- 프로그래밍 : 컴퓨터가 이해할 수 있는 언어로 실행 프로그램을 만드는 것.
- 프로그래밍 언어 : 컴퓨터가 이해할 수 있는 언어. 프로그래밍을 할 때 사용.



그림 5-1 프로그래밍 : 프로그래밍 언어로 프로그램을 작성하는 것

01. 프로그래밍 언어의 개요

II. 저급 언어와 고급 언어

■ 저급 언어

- **저급 언어(Low Level Language)** : 하드웨어 지향의 컴퓨터 내부 표현에 가까운 언어. 기계어와 어셈블리어로 구분됨.
 - 기계어 : 2진수 형태의 0과 1로 작성하며 컴퓨터가 직접 이해할 수 있는 언어.
 - 어셈블리어 : 기계어 명령을 간단한 기호로 표현한 언어.

```
0101 0001 0000 0000 0000 0111
0101 0001 0000 0000 0000 1000
0000 0000
0100 1000
0110 1001
```

(a) 기계어

```
CHAR0 0x0007, d
CHAR0 0x0008, d
STOP
.ASCII "Hi"
.END
```

(b) 어셈블리어

그림 5-2 저급 언어로 구현한 프로그램

01. 프로그래밍 언어의 개요

II. 저급 언어와 고급 언어

■ 고급 언어

- **고급 언어(High Level Language)** : 사람이 이해하기 쉬운 일상 언어와 기호를 사용해 프로그램을 작성할 수 있으며, 저급 언어보다 이식성이 높음.
- **대표적인 고급 언어** : C, 자바, 파이썬, 포트란, 코볼, 파스칼, C++ 등

```
#include <stdio.h>
```

```
int main() {  
    printf("Hi");  
    return 0;  
}
```

(a) C 언어

```
print("Hi")
```

(b) 파이썬

그림 5-3 고급 언어로 구현한 프로그램

01. 프로그래밍 언어의 개요

III. 프로그래밍 언어의 발전

■ 1세대 언어(1940년대)

- 전자식 컴퓨터가 개발되자 기계어를 이용한 프로그래밍이 시작됨.
- 기계어 프로그래밍의 단점을 해결하기 위해 만든 어셈블리어를 만듦.
- 어셈블리어는 기계어보다 작성하기 쉽지만, 광범위한 적용에는 한계가 있었음.

■ 2세대 언어(1950~60년대)

- 복잡한 프로그램을 작성하기에는 어셈블리어도 불편한 점이 있어 1950년대부터는 여러 고급 언어가 동시다발적으로 개발되기 시작함.
- 고급 언어를 기계어로 번역해 주는 컴파일러(Compiler)도 개발됨.

01. 프로그래밍 언어의 개요

III. 프로그래밍 언어의 발전

■ 3세대 언어(1970년대)

- 구조적 프로그래밍 기법을 따르는 파스칼(PASCAL)과 C 언어가 개발됨.
- 이러한 고급 언어를 바탕으로 유닉스와 MS-DOS 운영체제의 개발이 진행됨.

■ 4세대 언어(1980년대)

- 하드웨어의 가격이 내려가자 개인용 컴퓨터가 대중화되기 시작되었고, 더 크고 복잡한 프로그램을 개발하기 위해 객체 지향 프로그래밍 기법이 등장함.
- 다양한 프로그래밍 개발 툴이 지원돼, C++, 오브젝트-C, 펄(Perl) 등이 개발됨.

01. 프로그래밍 언어의 개요

III. 프로그래밍 언어의 발전

■ 5세대 언어(1990년대)

- GUI 환경 프로그래밍을 위한 여러 클래스의 라이브러리가 등장하면서 객체 지향 언어의 장점이 크게 주목받으면서 자바, 파이썬, 비주얼 베이직 등의 객체 지향 언어가 개발됨.
- 웹 프로그래밍이 보편화되면서 HTML, 자바스크립트 등의 언어가 개발됨.

■ 6세대 언어(2000년대)

- 2000년대에 들어서자 사용자는 더욱 간편하고 쉬운 방법으로 프로그래밍 작업을 하길 원해, 파워빌더(PowerBuilder), 델파이(Delphi) 등의 언어가 개발됨.
- 이외에도 C#, Go, 액션스크립트(ActionScript), 코틀린(Kotlin) 등이 개발됨.

01. 프로그래밍 언어의 개요

IV. 프로그래밍 언어별 특징

- **포트란(FORTRAN)** : 과학 계산용 언어로, 최근에도 우주항공, 기상 예측 등 복잡한 계산을 다루는 여러 분야에서 사용되고 있음.
- **코볼(COBOL)** : 회계 업무나 사무 자동화를 위해 개발된 프로그래밍 언어로, 영어 구문에 가까운 표기법을 가짐.
- **C** : 저급 언어와 고급 언어의 장점을 모두 가지고 있어 사용률이 높음. 또한 시스템 간 호환성이 좋고, 풍부한 연산자를 지원함. C를 바탕으로 유닉스, 리눅스 같은 운영체제가 개발될 수 있었으므로 컴퓨터 프로그래밍 환경의 성장을 이끈 매우 중요한 언어임.

01. 프로그래밍 언어의 개요

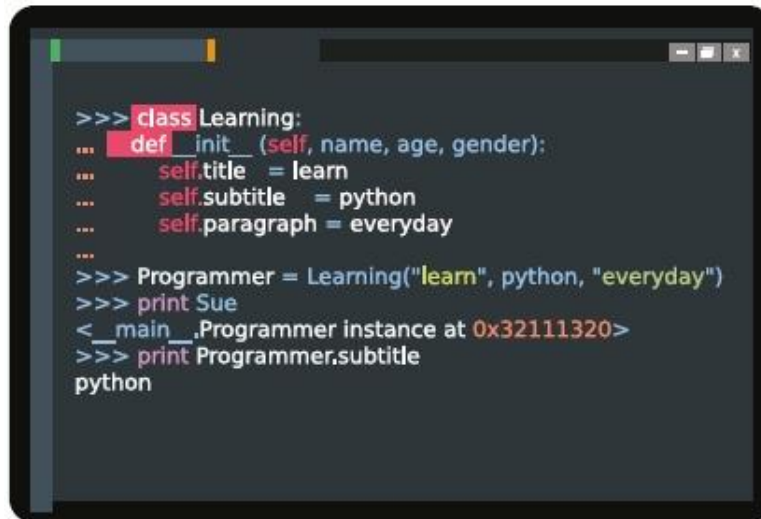
IV. 프로그래밍 언어별 특징

- **C++** : C의 유연성에 객체 지향 요소를 추가한 언어로, 규모가 큰 응용 프로그램을 제작할 때 많이 사용됨.
- **C#** : 웹 환경에서 C++가 가지는 제한적 요소를 해결하기 위해 마이크로소프트사가 개발한 언어로, C++에 기반을 둔 객체 지향 언어.
- **자바(Java)** : 썬 마이크로시스템즈사에서 만든 객체 지향 언어로, C++가 가진 기능 대부분은 유지하면서 복잡성 등의 단점은 개선함. 컴퓨터나 운영체제 종류에 구애받지 않는 이식성이 장점이지만, 수행 속도가 느리다는 단점도 있음.

01. 프로그래밍 언어의 개요

IV. 프로그래밍 언어별 특징

- 파이썬(Python) : 인터프리터 방식의 프로그래밍 언어로, 객체 지향 언어 중 하나임. 문법 구조가 간단해 교육용 프로그래밍 언어로 인기가 많음. 또한 데이터 과학에 관한 다양한 라이브러리를 제공해 사용률이 높아지고 있음.
 - 파이썬의 특징 : 무료 제공, 배우기 쉬운 언어, 인터프리터 언어, 객체 지향 언어, 동적 타이핑 언어, 다양한 라이브러리 제공.



```
>>> class Learning:
...     def __init__(self, name, age, gender):
...         self.title = learn
...         self.subtitle = python
...         self.paragraph = everyday
...
>>> Programmer = Learning("learn", python, "everyday")
>>> print Programmer
<__main__.Programmer instance at 0x32111320>
>>> print Programmer.subtitle
python
```

그림 5-4 파이썬 코드



[파이썬은 어디에 쓰일까?\(04:09\)](#)

02

프로그래밍 언어의 구현 기법

02. 프로그래밍 언어의 구현 방법

I. 소스코드 작성

- 프로그래밍 언어를 이용해 프로그램을 구현하려면 고급 언어를 선택하고 소스코드를 작성해, 소스파일을 만들어야 함.

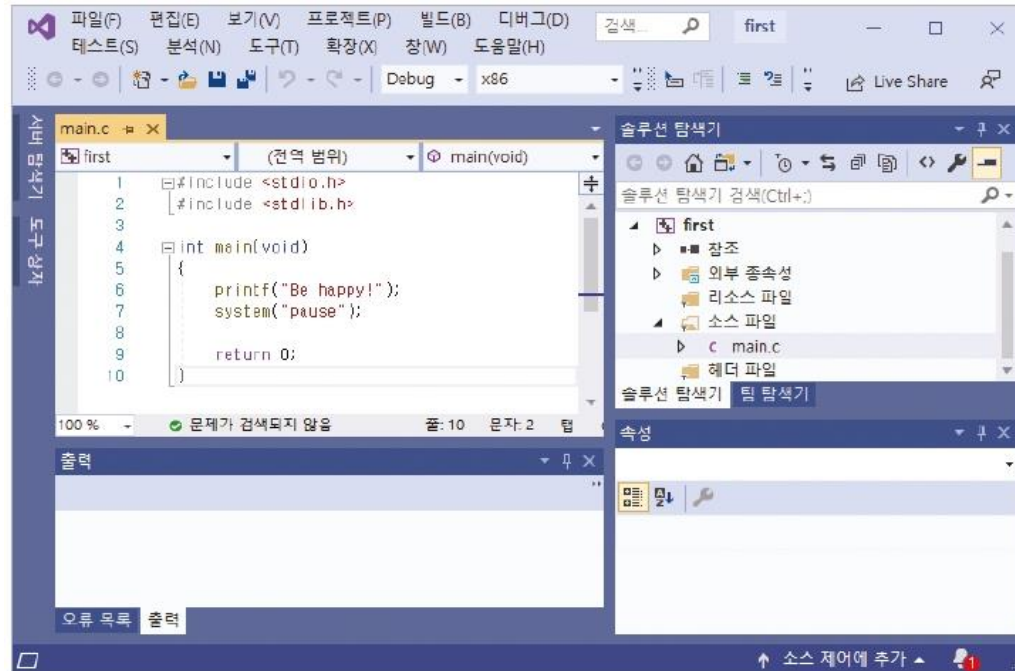


그림 5-5 통합개발환경 Visual Studio 2019를 이용한 소스코드 작성(C 언어)

02. 프로그래밍 언어의 구현 방법

II. 프로그래밍 언어의 번역

- 프로그래밍 언어의 번역 : 고급 프로그래밍 언어의 소스코드를 기계어로 번역하는 것으로, 컴파일러 방식과 인터프리터 방식으로 나뉜다.

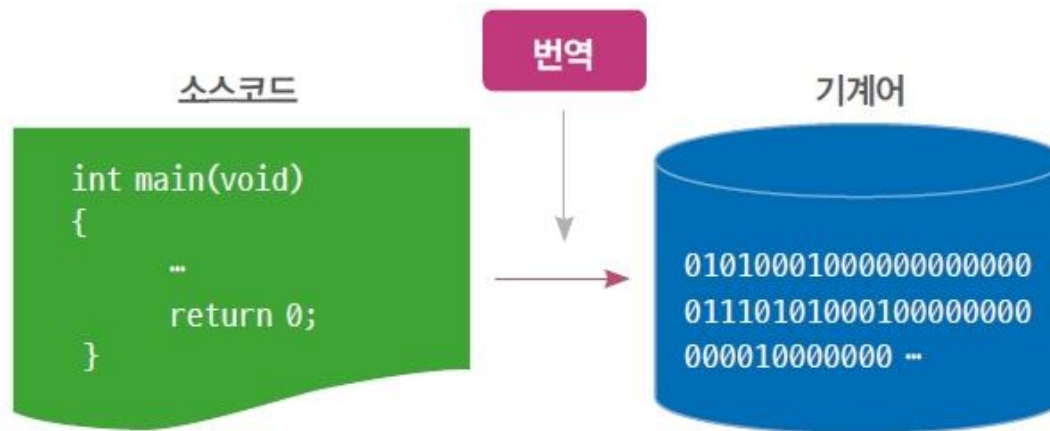


그림 5-6 프로그래밍 언어의 번역(소스코드→기계어)

02. 프로그래밍 언어의 구현 방법

II. 프로그래밍 언어의 번역

■ 컴파일러

- **컴파일러(Compiler)** : 프로그램 전체 소스코드를 기계어로 한번에 번역해 실행 파일을 만든 뒤 프로그램을 실행하는 방식으로, C, 자바 등이 컴파일러 방식을 사용함.
- **컴파일(Compile)** : 소스 전체를 개체 파일로 번역하는 과정.

■ 인터프리터

- **인터프리터(Interpreter)** : 소스코드를 한 행씩 읽어 가며 번역과 실행을 동시에 수행하는 방식으로, 프로그램이 실행 중일 때 고급 언어로 작성된 소스코드 명령어를 하나씩 번역하는 것.
- 컴파일러를 이용한 방식보다 속도가 느리지만, 즉각적인 피드백이 가능.



[파이썬이 왜 C보다 느린가? 컴파일과 인터프리터의 기초 개념\(05:29\)](#)

02. 프로그래밍 언어의 구현 방법

II. 프로그래밍 언어의 번역

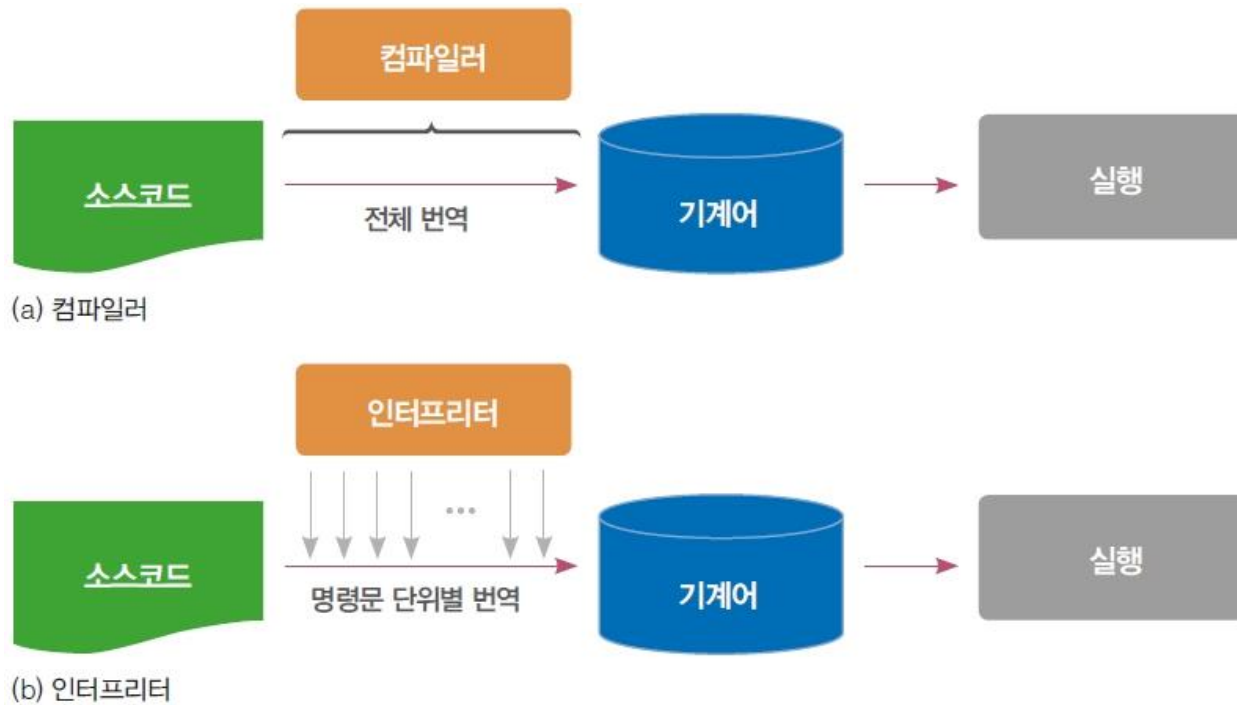


그림 5-7 컴파일러와 인터프리터 진행 과정

02. 프로그래밍 언어의 구현 방법

III. 프로그래밍 언어의 전체 실행 과정

- ① 사용자는 프로그래밍 언어로 소스코드를 작성해 소스파일을 만들
- ② 이 소스파일은 번역을 거쳐 개체 파일이 됨.
- ③ 개체 파일은 링커 프로그램을 통해 링킹 과정을 거침.
- ④ 이렇게 작성된 프로그램은 컴퓨터 메인메모리에 로드되는 로더 과정을 거침.
- ⑤ 모든 과정이 마무리되면 프로그램이 실행됨.

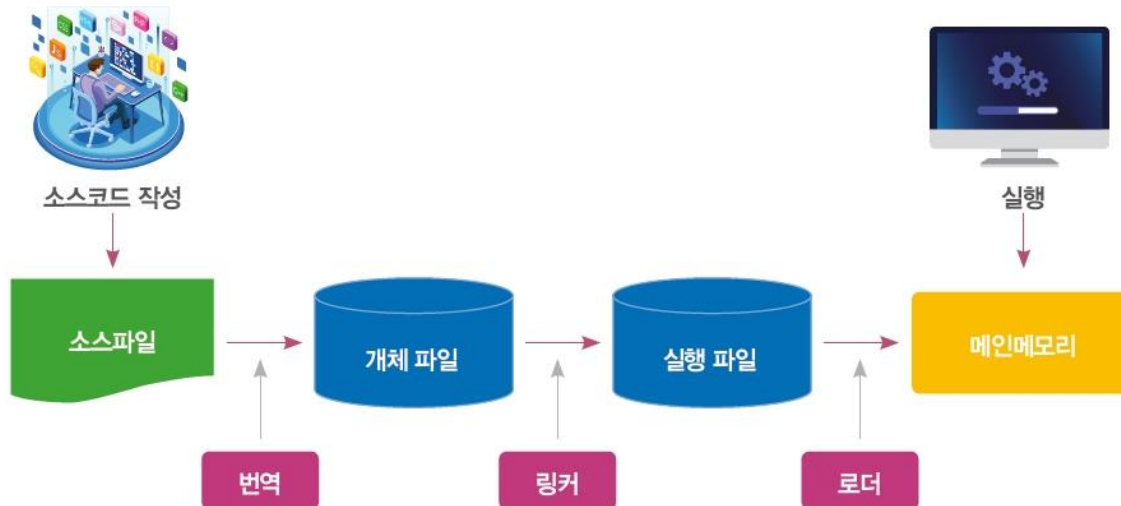


그림 5-8 프로그래밍 언어의 전체 실행 과정

03

프로그래밍 언어의 분류

03. 프로그래밍 언어의 분류

I. 절차 지향 언어

- **절차 지향 언어(Procedure-oriented Language)** : 프로그램을 작성할 때 실행 순서를 중심으로 설계하는 프로그램 작성 언어로, C, C++, 포트란, 베이직, 코볼 등이 있음.

코드 5-1 절차 지향 언어의 예 : C

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 6;
06     int b = 15;
07     printf("a + b = %d\n", a + b);
08
09     return 0;
10 }
```

출력 결과

a + b =21

03. 프로그래밍 언어의 분류

II. 논리형 언어

- **논리형 언어(Logical Language)** : 논리학의 관계식 형태로 프로그램을 기술한 언어로, 프롤로그(PROLOG)가 대표적임. 논리식을 바탕으로 객체 간의 관계에 대한 문제를 해결할 때 주로 사용.

코드 5-2 논리형 언어의 예 : 프롤로그

```
01 father(X,Y) :- parent(X,Y), male(X).
02 mother(X,Y) :- parent(X,Y), female(X).
03 sibling(X,Y) :- parent(Z,X), parent(Z,Y).
04
05 parent(trude, sally).
06 parent(tom, sally).
07 parent(tom, erica).
08 parent(mike, tom).
09 male(tom).
10 female(trude).
11 male(mike).
```

출력 결과

```
>> sibling(sally, erica).
yes.
```

03. 프로그래밍 언어의 분류

III. 객체 지향 언어

 [객체 지향 프로그래밍이 필요한 이유\(05:02\)](#)

- 객체 지향 언어(Object-oriented Language) : 프로그램을 독립된 단위인 '객체'들의 집합으로 파악하는 언어로, 자바, 파이썬, C++, C# 등이 있음.

코드 5-3 객체 지향 언어의 예 : 파이썬

```
01 class car:
02     def __init__(self):
03         self.speed = 0
04
05     def print_speed(self):
06         print("Speed is ", self.speed)
07
08     def speed_up(self):
09         self.speed += 1
10
11 myCar = car()
12 myCar.print_speed()
13 myCar.speed_up()
14 myCar.print_speed()
```

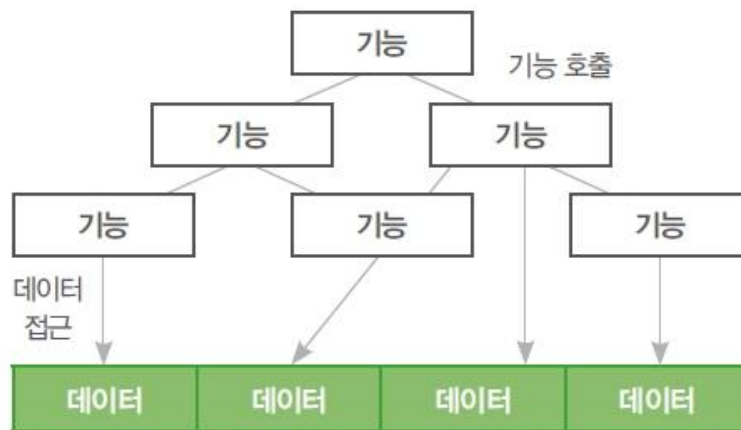
출력 결과

```
Speed is 0
Speed is 1
```

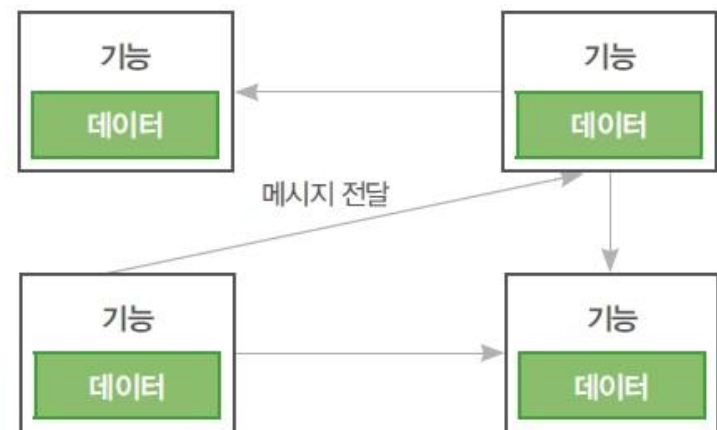
03. 프로그래밍 언어의 분류

III. 객체 지향 언어

- 절차 지향 언어는 데이터와 기능을 별도로 관리해 기능을 호출한 뒤 데이터에 접근해 일을 처리하는데 반해, 객체 지향 언어는 기능과 데이터를 하나로 묶어 캡슐화한 후 메시지를 전달해 일을 처리하므로 훨씬 효율적임.



(a) 절차 지향 언어



(b) 객체 지향 언어

그림 5-9 절차 지향 언어와 객체 지향 언어의 차이점

03. 프로그래밍 언어의 분류

IV. 함수형 언어

- **함수형 언어(Functional Language)** : 데이터에 대한 함수 적용을 바탕으로 처리 과정을 기술하는 언어로, 대표적인 언어로는 리스프(LISP), 스킴(Scheme) 등이 있음.

코드 5-4 함수형 언어의 예 : 파이썬

```
01 def sum_lst(lst):  
02     if not lst:  
03         return 0  
04     else:  
05         return lst[0] + sum_lst(lst[1:])
```

04

프로그래밍 문법

04. 프로그래밍 문법

I. 변수

- **변수(Variable)** : 어떠한 값을 저장하는 메모리 공간.
- **변수의 형식** : 정수형(Integer), 실수형(Float), 문자열형(Char) 등
- 변수 선언은 일종의 그릇을 준비하는 것과 같은데 파이썬은 변수를 따로 선언하지 않아도 됨. 이를 동적 타이핑이라고 함.
- 파이썬과 같은 동적 타이핑 언어는 코드를 작성할 때 시간이 단축돼 생산성이 높지만, 추후 유지보수를 해야 할 땐 정적 타이핑 언어보다 가독성이 떨어진다는 단점이 있음.

04. 프로그래밍 문법

I. 변수

코드 5-5 다양한 변수 선언(정수, 실수, 문자열)

```
01 a = 33
02 b = 10.2
03 professor = 'keechul'
04 print(a)
05 print(b)
06 print(professor)
```

출력 결과

```
33
10.2
keechul
```

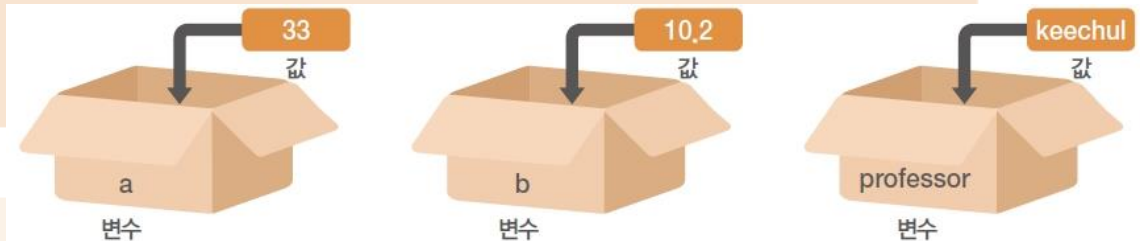


그림 5-10 변수와 값

- [코드 5-5] a는 정수형 변수, b는 실수형 변수, professor는 문자열형 변수로, 기호 '='는 대입 연산자로, 오른쪽 값을 왼쪽의 변수에 대입한다는 뜻.

04. 프로그래밍 문법

I. 변수

- 변수 이름을 지을 때 지켜야 할 규칙
 - 알파벳, 숫자, 밑줄(_)로 선언할 수 있음.
ex) `data = 0, _a12 = 2, _gg = ' afd'`
 - 변수명은 의미 있는 단어로 표기하는 것이 좋음.
ex) `professor_name = ' keechul'`
 - 변수명은 대·소문자를 구분해야 함.
ex) `ABC`와 `abc`는 같은 단어가 아님
 - 특별한 의미가 있는 예약어는 사용할 수 없음.
ex) `for`, `if`, `else` 등의 데이터

04. 프로그래밍 문법

I. 변수

하나 더 알기 입력문(input)과 출력문(print)

- **print()** : 출력문을 나타내는 함수.

코드 5-6 print()

```
01 print("Hello World!")
02 print(1)
03 print(3+7)
04 print("sum is:", 3+7)
```

출력 결과

```
Hello World!
1
10
sum is: 10
```

04. 프로그래밍 문법

I. 변수

하나 더 알기 입력문(input)과 출력문(print)

- **input()** : 입력문을 나타내는 함수.

코드 5-7 input()

```
01 a = input('Please, input a number: ')\n02 print('You entered:', a)
```

출력 결과

```
Please, input a number: 1\nYou entered: 1
```

04. 프로그래밍 문법

II. 자료형

- **자료형(Data Type)** : 프로그램에서 사용되는 자료의 구조를 정의한 것.

표 5-1 자료형 기호

자료형	설명	예
int	정수형(Integer)	5, 10, 100, ...
float	실수형(Float)	3.14, 1.68, ...
bool	불린형(Boolean)	True, False
str	문자열형(String)	'Hello World'

04. 프로그래밍 문법

II. 자료형

- 파이썬은 변수를 사용할 때 자료형을 지정하는 선언 과정이 없는 대신, 프로그램 실행 중 할당되는 값에 맞춰 자료형을 판단.

코드 5-8 자료형

```
01 a = 1
02 b = 1.0
03 sum = a + b
04 print(type(a))
05 print(type(b))
06 print(type(sum))
```

출력 결과

```
<class 'int'>
<class 'float'>
<class 'float'>
```

04. 프로그래밍 문법

III. 연산자

- 파이썬은 산술연산자, 비교연산자, 논리연산자를 제공.
- 산술연산자** : 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/), 지수 승(**), 몫(/), 나머지(%)

코드 5-9 산술연산자

```
01 print(3+5, 9-2, 3*8, 6/4)
02 print(3**4)                # 지수 승 연산
03 print(14//4)               # 몫 연산
04 print(8%3)                 # 나머지 연산
```

출력 결과

```
8 7 24 1.5
81
3
2
```

04. 프로그래밍 문법

III. 연산자

- 논리연산자와 비교연산자

표 5-2 논리연산자와 비교연산자 기호

논리연산자	기호	설명
	and	둘 다 참일 때 참
	or	하나만 참이어도 참
	not	논리 상태 반전

비교연산자	기호	설명	기호	설명
	==	같다	<	왼쪽이 오른쪽보다 작다
	!=	같지 않다	>=	왼쪽이 오른쪽보다 크거나 같다
	>	왼쪽이 오른쪽보다 크다	<=	왼쪽이 오른쪽보다 작거나 같다

04. 프로그래밍 문법

III. 연산자

- 논리연산자와 비교연산자

코드 5-10 비교연산자와 논리연산자

```
01 print(1==1 and 2!=3)    # 1은 1과 같고, 2는 3과 같지 않다.  
02 print(2>1 or 3<2)      # 2는 1보다 크다. 3은 2보다 작다. 둘 중 하나는 맞다.  
03 print(not 10 >= 5)      # 10은 5보다 크거나 같지 않다.
```

출력 결과

```
True  
True  
False
```

04. 프로그래밍 문법

IV. 조건문

- **조건문** : 조건에 따라 특정 동작을 하게 만드는 프로그래밍 명령어.

▪ if문

- **if문** : 가장 단순한 형태의 명령어로, 참일 때는 명령을 실행하고, 거짓일 때는 아무 것도 실행하지 않음.

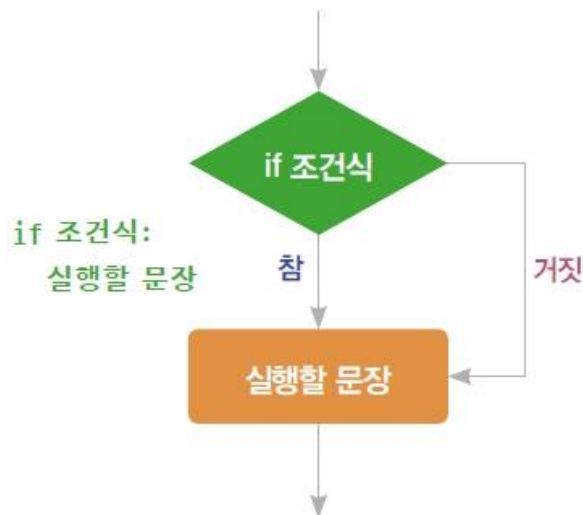


그림 5-11 if문의 형식과 순서도

04. 프로그래밍 문법

IV. 조건문

▪ if문

코드 5-11 if문

```
01 a = 10
02 b = 10
03 if a == b:
04     print(a, 'is same to', b)
```

출력 결과

```
10 is same to 10
```

- [코드 5-11]에서 a와 b가 같다는 것은 참이므로 '10 is same to 10' 결과 값을 출력.
- 만약 if문이 거짓이면 아무것도 출력되지 않음.
- if문 끝에는 콜론(:)을 꼭 붙여야 하며, if문 다음에 등장하는 실행할 문장은 반드시 들여쓰기해야 함.

04. 프로그래밍 문법

IV. 조건문

▪ if문

하나 더 알기 들여쓰기

- 파이썬이 C나 자바와 다른 점 중 하나가 바로 들여쓰기 기능.
- 파이썬에서 사용하는 들여쓰기는 문법적인 요소로, 코드들의 계층적인 정보를 담고 있으므로 반드시 지켜져야 함.
- 들여쓰기하지 않고 넣으면, 에러 메시지인 'SyntaxError'만 뜨고 결과는 출력되지 않음.

04. 프로그래밍 문법

IV. 조건문

▪ if-else문

- **if-else문** : 참일 때 실행할 문장과 거짓일 때 실행할 문장을 다르게 적용해야 할 때 사용하는 명령어.



그림 5-12 if-else문의 형식과 순서도

04. 프로그래밍 문법

IV. 조건문

▪ if-else문

코드 5-12 if-else문

```
01 a = 20
02 b = 10
03 if a > b:
04     print("max is", a)
05 else:
06     print("max is", b)
```

출력 결과

```
max is 20
```

- [코드 5-12]는 변수 a와 b 값을 비교해 a가 b보다 크면 a를 출력하고, a가 b보다 작거나 같으면 b를 출력함.

04. 프로그래밍 문법

IV. 조건문

▪ if-elif-else문

- if-elif-else문 : 조건이 여러 개일 때 사용하는 명령어.

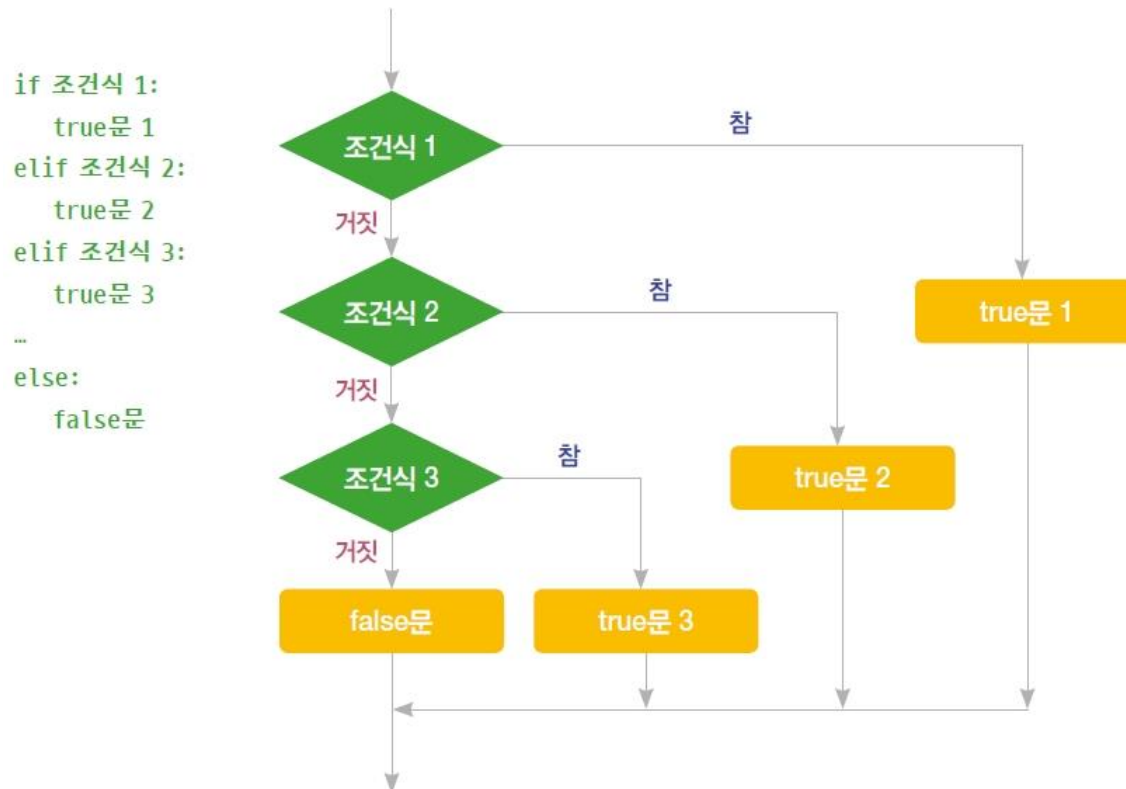


그림 5-13 중첩 if문의 형식과 순서도

04. 프로그래밍 문법

IV. 조건문

▪ if-elif-else문

- 중첩 if문을 간단히 표현하는 방식인 if-elif-else문을 많이 사용함.

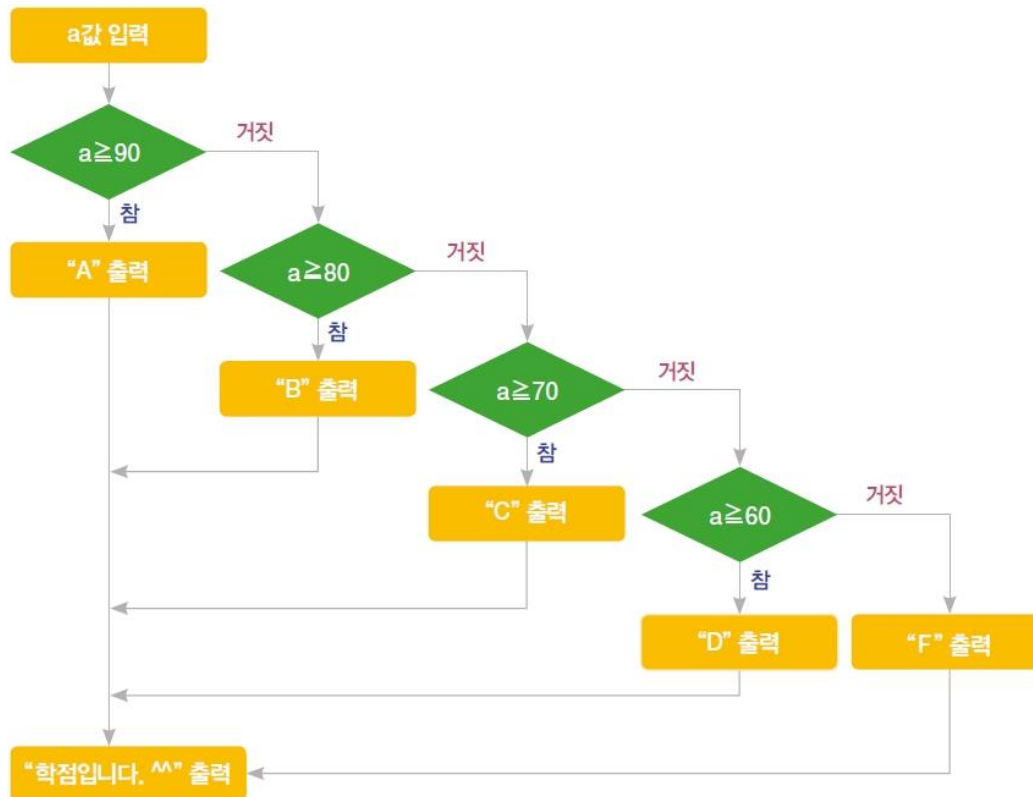


그림 5-14 if-elif-else문의 적용 예: 학점 계산 순서도

04. 프로그래밍 문법

IV. 조건문

▪ if-elif-else문

- [그림 5-14]를 파이썬 코드로 나타내면 [코드 5-13]과 같음.

코드 5-13 if-elif-else문

```
01 score = int(input("Enter your score: "))
02
03 if score >= 90:                # 90 이상일 경우 A
04     grade = 'A'
05 elif score >= 80:             # 80 이상일 경우 B
06     grade = 'B'
07 elif score >= 70:             # 70 이상일 경우 C
08     grade = 'C'
09 elif score >= 60:             # 60 이상일 경우 D
10     grade = 'D'
11 else: grade = 'F'              # 모든 조건에 만족하지 못할 경우 F
12
13 print(grade)
```

출력 결과

```
Enter your score: 98          ← 사용자 점수 입력
A                             ← 학점 출력
```

04. 프로그래밍 문법

V. 반복문

- **반복문(Repetitive Statement)** : 소스코드 내에서 특정 부분의 코드가 반복적으로 수행되게 하는 명령어임. 컴퓨터 프로그램은 대부분 자동화된 계산에 사용되므로 반복문이 자주 사용됨.
- **파이썬에서 제공하는 반복문** : for문과 while문

▪ for문

- **for문** : 기본적인 반복문으로, 범위를 지정하여 동작을 반복 수행함.

04. 프로그래밍 문법

V. 반복문

▪ for문

- [코드 5-14]는 for문의 예시. range함수는 다음과 같은 기본 문법 구조를 가지며 증가 값을 생략할 수 있음.

• for 변수 in range(시작 값, 끝 값+1, 증가 값)

코드 5-14 for문

```
01 for loopier in range(2, 6):  
02     print(loopier)
```

출력 결과

```
2  
3  
4  
5
```

04. 프로그래밍 문법

V. 반복문

▪ while문

- **while문** : 조건식이 참일 때만 동작을 반복하는 명령어.

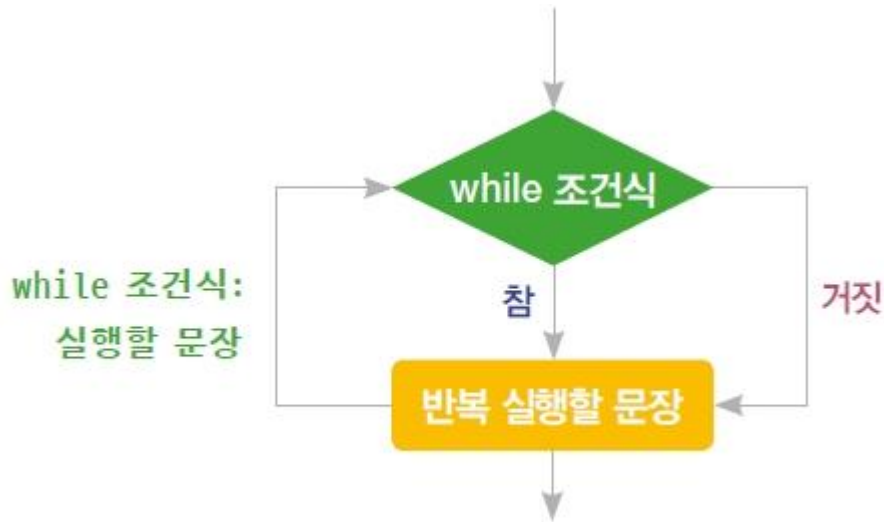


그림 5-15 while문의 형식과 순서도

04. 프로그래밍 문법

V. 반복문

▪ while문

- [코드 5-15]는 '변수 i를 1부터 10까지 1씩 증가시키며 결과 값을 출력하라'는 코드.

코드 5-15 while문 I

```
01 i = 1
02
03 while i <= 10:
04     print(i)
05     i = i + 1
```

출력 결과

```
1
2
3
...
10
```

04. 프로그래밍 문법

V. 반복문

▪ while문

- [코드 5-16]은 while문을 이용해 1부터 10까지의 합을 결과 값으로 출력하는 코드.

코드 5-16 while문 II

```
01 sum = 0
02 i = 1
03
04 while i <= 10:
05     sum = sum + i
06     i = i + 1
07
08 print("sum from 1 to 10 is", sum)
```

출력 결과

```
sum from 1 to 10 is 55
```

04. 프로그래밍 문법

V. 반복문

▪ 반복문의 제어

- **break문** : 작업 중간에 반복 동작을 종료하고 싶을 때 사용하는 명령어.

코드 5-17 break문

```
01 for i in range(10):  
02     if i == 3:  
03         break                # i가 3이 되면 반복 동작 종료  
04     print(i)  
05 print("End of Program")      # 반복 동작 종료 후 'End of Program' 출력
```

출력 결과

```
0  
1  
2  
End of Program
```

04. 프로그래밍 문법

V. 반복문

▪ 반복문의 제어

- **continue문** : 특정 조건이 되면 명령을 건너뛰고 다음 반복문을 수행하는 명령어.

코드 5-18 continue문

```
01 for i in range(10):
02     if i == 5:
03         continue          # i가 6인 상태에서 다시 계산 수행
04     print(i)
05 print("End of Program")    # 반복 동작 종료 후 'End of Program' 출력
```

출력 결과

```
0
1
2
3
4
6
7
8
9
End of Program
```