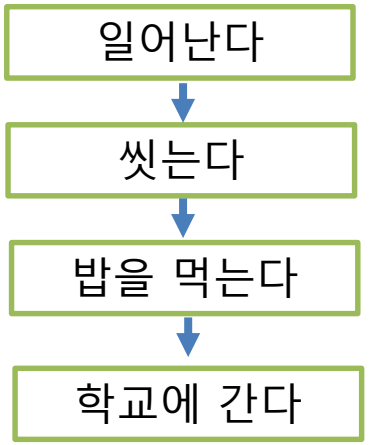
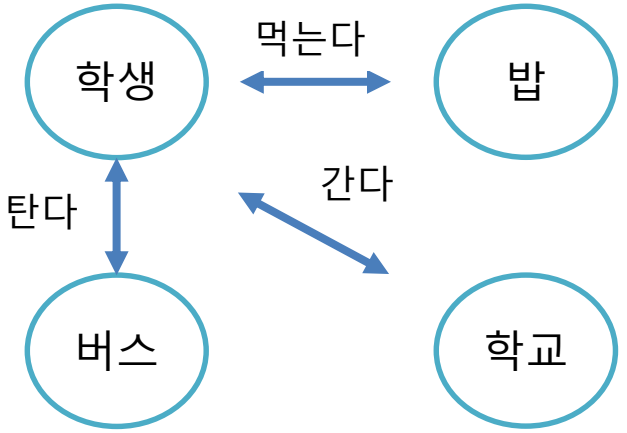
8장. 클래스와 상속

객체 지향 프로그래밍

- **객체(Object)란?**
  - "의사나 행위가 미치는 대상" -> 사전적 의미
  - 구체적, 추상적 데이터 단위 (구체적- 책상, 추상적-회사)
- **객체지향 프로그래밍(Objected Oriented Programming)**
  - 객체를 기반으로 하는 프로그래밍
  - 먼저 객체를 만들고 객체 사이에 일어나는 일을 구현함.



<절차지향 –C언어>



<객체지향 – Python>

클래스(class)

- **클래스란?**
  - ✓ **객체에 대한 속성과 기능을 코드로 구현** 한 것
  - ✓ 객체에 대한 설계도 또는 청사진.
- **객체의 속성과 기능**
  - ✓ 객체의 특성(property), 속성(attribute) -> **멤버 변수**
  - ✓ 객체가 하는 기능 -> **멤버 함수**

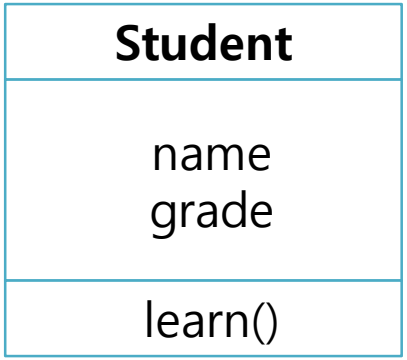
```
class 클래스 이름 :
    def __init__(self):
        멤버변수

    def 함수이름(self):
        return
```

학생 클래스

- 속성(멤버변수) : 학번, 이름, 학년, 사는 곳 등..
- 기능(함수) : 수강신청, 수업듣기, 시험 보기 등..

클래스(class) 정의

- **학생 클래스 정의 및 사용**



객체(인스턴스) = 클래스()

객체.멤버변수
객체.멤버 메서드

객체를 생성한 후 점(.) 연산자를 사용하여 멤버변수나 메서드에 접근함

```
class Student:
    name = "김하나"
    grade = 5

st1 = Student()
print(st1.name, st1.grade)
```

클래스(class) 정의

- **생성자(constructor) – 기본 생성자**
  - 클래스를 생성할 때 호출되는 명령어 집합, 초기자라고도 한다.
  - 생성자는 **__init__()**의 형태로 작성하고, 리턴값이 없다
  - 클래스 내의 모든 함수(메서드)의 매개변수에 **self**를 넣어줌

```
class Student:
    def __init__(self):
        self.name = "콩쥐"
        self.grade = 1
        print("생성자")

    def learn(self):
        print("수업을 듣습니다.")

s = Student() # 객체 생성
print(s)  # 객체 출력
print(type(s)) # 자료형 : 클래스
print(s.name, "학생은", s.grade, "학년입니다.")
s.learn()
```

```
생성자
<__main__.Student object at 0x000001A1D2336A50>
<class '__main__.Student'>
콩쥐 학생은 1학년입니다.
수업을 듣습니다.
```

클래스(class) 정의

- **생성자(constructor) – 매개변수가 있는 생성자**
  - 객체를 생성할 때 매개 변수로 전달하여 멤버변수에 저장함

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def learn(self):
        print("수업을 듣습니다.")

s1 = Student("김하나", 1)
print(f"{s1.name} 학생은 {s1.grade}학년입니다.")
s1.learn()

s2 = Student("이돌", 3)
print(f'{s2.name} 학생은 {s2.grade}학년입니다.')
s2.learn()
```

__str__(self) : 객체 정보 함수

- **__str(self)__ 사용하기**

  문자열을 return하는 함수이다. 객체의 정보를 담고 있다.

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def learn(self):
        print("수업을 듣습니다.")

    def __str__(self):
        return f"{self.name} 학생은 {self.grade}학년입니다."

s1 = Student("김하나", 1)
print(s1)
s1.learn()

s2 = Student("이돌", 3)
print(s2)
s2.learn()
```

객체 리스트

- **학생 리스트 만들기**

```
***** 학생 명단 *****
김하나 학생은 1학년입니다.
박열 학생은 3학년입니다.
이넷 학생은 2학년입니다.
```

```
students = [
    Student("김하나", 1),
    Student("박열", 3),
    Student("이넷", 2)
]

print("***** 학생 명단 *****")
for st in students:
    print(st)
```

계산기 클래스 만들기

- 계산기 클래스 만들기

Calculator
x, y
add() sub() mul() div()

```
class Calculator:
    def __init__(self):
        self.x = 0

    def add(self, y): # 덧셈
        self.x += y
        return self.x

    def sub(self, y): # 뺄셈
        self.x -= y
        return self.x

    def mul(self, y): # 곱셈
        self.x *= y
        return self.x
```

계산기 클래스 만들기

- 계산기 클래스 만들기

```
    def div(self, y): # 나눗셈
        if y != 0:
            self.x /= y
        else:
            print("Error: 0으로 나눌 수 없습니다.")
        return self.x

cal = Calculator()
print(cal.add(10))   #10
print(cal.sub(4))    #6
print(cal.mul(2))    #12
# print(cal.div(10)) #1.2
# print(cal.div(0))  #12
```

정보 은닉(Information Hiding)

- **정보 은닉**
  - 멤버 변수에 언더스코어(__) 2개를 붙이면 직접 접근할 수 없음
  - **메서드(getter, setter) : get + 변수 이름(), set + 변수이름()**

접근 제어	설 명
public	외부 클래스 어디에서나 접근 할수 있다.
private	같은 클래스 내부 가능, 그 외 접근 불가

정보 은닉(Information Hiding)

- 정보 은닉(접근 제한)

```
class BankAccount:
    def __init__(self):
        self.__ano = ""
        self.__owner = ""
        self.__balance = ""

account1 = BankAccount()
print(account1.__ano)
```

```
Traceback (most recent call last):
  File "d:\korea_IT\pyworks\classes\클래스.py", line 121, in <module>
    print(account1.__ano)
          ^^^^^^^^^^^^^^
AttributeError: 'BankAccount' object has no attribute '__ano'
```

정보 은닉(Information Hiding)

- 은행 계좌 만들기

```
class BankAccount:
    def __init__(self):
        self.__ano = ""
        self.__owner = ""
        self.__balance = ""

    # 계좌 번호
    def set_ano(self, ano):
        self.__ano = ano

    def get_ano(self):
        return self.__ano

    # 계좌주
    def set_owner(self, owner):
        self.__owner = owner

    def get_owner(self):
        return self.__owner
```

```
계좌번호: 12-1234
계좌주: 김기용
잔고: 20000
```

정보 은닉(Information Hiding)

- 은행 계좌 만들기

```
    # 잔고
    def set_balance(self, balance):
        self.__balance = balance

    def get_balance(self):
        return self.__balance

account1 = BankAccount()
# setter
account1.set_ano("12-1234")
account1.set_owner("김기용")
account1.set_balance(20000)

# getter
print("계좌번호:", account1.get_ano())
print("계좌주:", account1.get_owner())
print("잔고:", account1.get_balance())
```

클래스 변수

◆ 클래스 변수

- 해당 클래스를 사용하는 모두에게 공용으로 사용되는 변수.
- 생성자 def __init__() 위에 위치
- 클래스 이름으로 직접 접근함

```
class Dog:
    kind = "진돗개"  #클래스 변수

    def __init__(self, name):
        self.name = name

dog1 = Dog("백구")
dog2 = Dog("밀크")
```

```
print(dog1.name)  # dog1만 유일
print(dog2.name)  # dog2만 유일

# 모든 dog이 공유
# print(dog1.kind)
# print(dog2.kind)

# 클래스 이름으로 직접 접근(올바른 유형)
print(Dog.kind)
```

인스턴스 변수& 클래스 변수

◆ 카운터 만들기

인스턴스 변수

```
class Counter:
    def __init__(self):
        self.x = 0
        self.x += 1

    def get_count(self):
        return self.x

c1 = Counter()
print(c1.get_count())    # 1
c2 = Counter()
print(c2.get_count())    # 1
c3 = Counter()
print(c3.get_count())    # 1
```

클래스 변수

```
class Counter:
    x = 0    # 클래스 변수

    def __init__(self):
        Counter.x += 1
        # 클래스이름으로 직접 접근

    def get_count(self):
        return self.x

c1 = Counter()
print(c1.get_count())    # 1
c2 = Counter()
print(c2.get_count())    # 2
c3 = Counter()
print(c3.get_count())    # 3
```

클래스(class) 모듈 사용

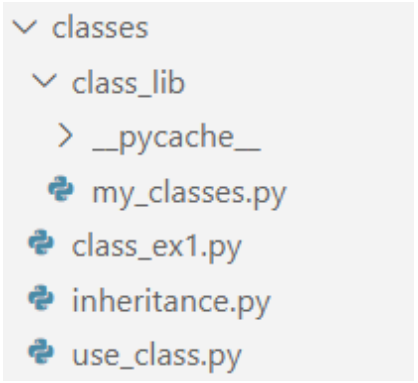
- 클래스를 모듈로 import 하는 방법 – 외부 파일에서 사용

```python
class Student:
    def __init__(self, name, grade):
        self.name = name    #인스턴스 변수
        self.grade = grade

    def learn(self):
        print("수업을 듣습니다.")

    def __str__(self):  #정보 출력 메서드
        return f"{self.name} 학생은 {self.grade}학년입니다."
```

```python
# 외부에서 사용할 때 실행 방지
if __name__ == "__main__":
    # Student 인스턴스 생성
    s1 = Student("김하나", 1)
    print(s1)
    s1.learn()

    s2 = Student("이돌", 3)
    print(s2)
    s2.learn()
```

클래스(class) 모듈 사용

- 모듈 - 클래스 사용하기




```
from class_lib.my_classes import Student

st1 = Student("흥부", 1)
print(st1)
st1.learn()

st2 = Student("놀부", 3)
print(st2)
st2.learn()
```

```
흥부 학생은 1학년입니다.
수업을 듣습니다.
놀부 학생은 3학년입니다.
수업을 듣습니다.
```

쇼핑몰 장바구니 클래스

- 쇼핑몰 장바구니 구현하기

```
class Cart:
    def __init__(self, user):
        self.user = user
        self.items = []  #장바구니 리스트

    def add(self, *goods):  # 여러 상품 한 번에 추가(가변 매개변수)
        self.items.extend(goods)

    def remove(self, item):
        if item in self.items:
            self.items.remove(item)

    def __str__(self):
        return f"{self.user}'s 장바구니: {self.items}"
```

쇼핑몰 장바구니 클래스

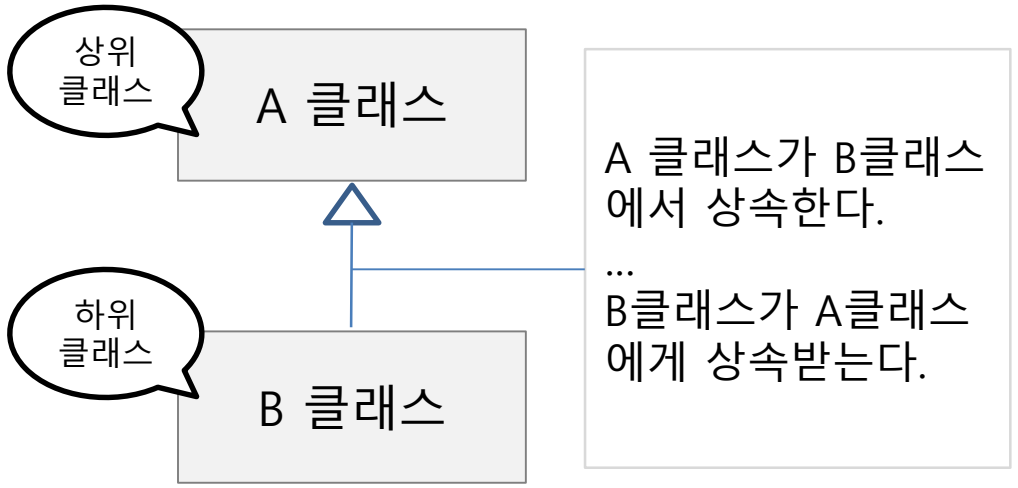
- 쇼핑몰 장바구니 구현하기

```
# 사용
my_cart = Cart("장그래")
my_cart.add("계란", "우유", "라면")
my_cart.remove("우유")
print(my_cart)  # "장그래's 장바구니: ['계란', '라면']"
```

상속(Inheritance)

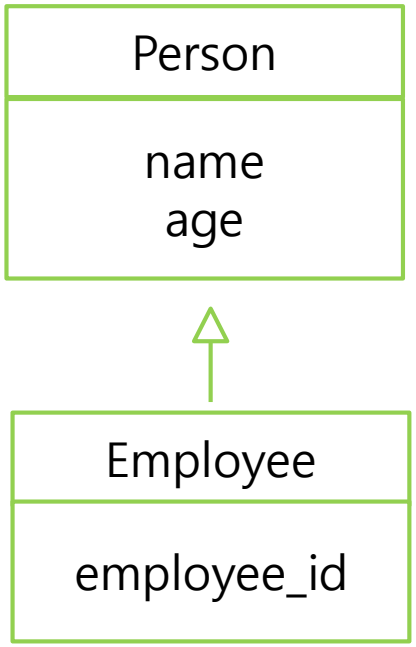
- **상속이란?**
  - 클래스를 정의할때 이미 구현된 클래스를 상속(inheritance) 받아서 속성이나 기능이 확장되는 클래스를 구현함.
  - 클래스 상속 문법

```
class 클래스 이름(상속할 클래스 이름)
```



상속(inheritance)

- 클래스 상속



부모 클래스(사람)

고유 속성 – name, age

자식 클래스(사원)

고유 속성 – department

상속(inheritance)

- **Employee 클래스 – Person 클래스 상속**

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_info(self):
        return f"{self.name}({self.age})"
```

```
# Person 클래스 상속
class Employee(Person):
    pass

e1 = Employee("김산", 31)
print(e1.get_info())
```

상속(inheritance)

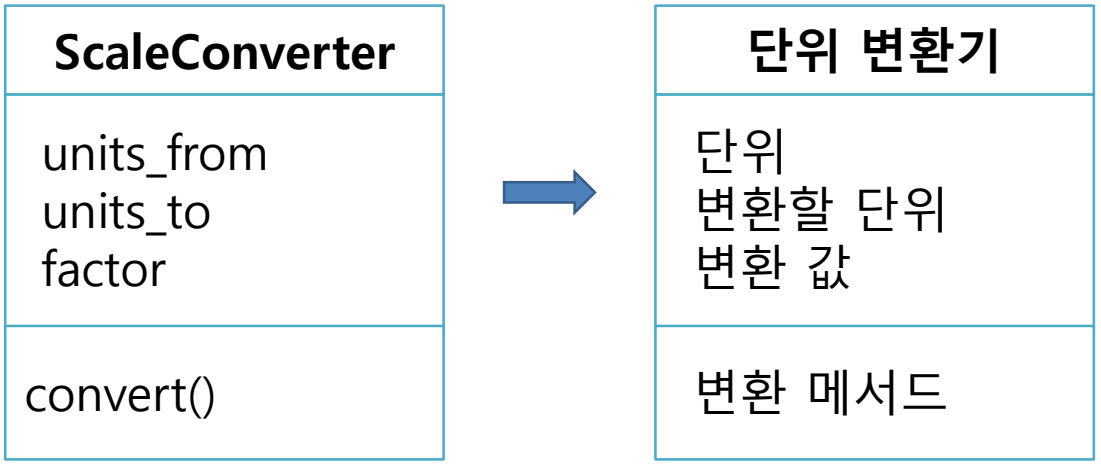
- **super() 사용 – 생성자 상속**

```
# 직원 ID 속성 추가
class Employee(Person):
    def __init__(self, name, age, department):
        super().__init__(name, age) #부모 클래스 생성자 호출
        self.department = department #직원 고유 속성

    def get_info(self): #메서드 재정의(오버라이드)
        return f"{self.name}({self.age}) - {self.department}"

e2 = Employee("이강", 26, "전산실")
print(e2.get_info())
```

```
김산(31)
이강(26) - 전산실
```

상속(inheritance)

- **단위변환- inch(인치)를 mm로 변환하는 클래스**



상속(inheritance)

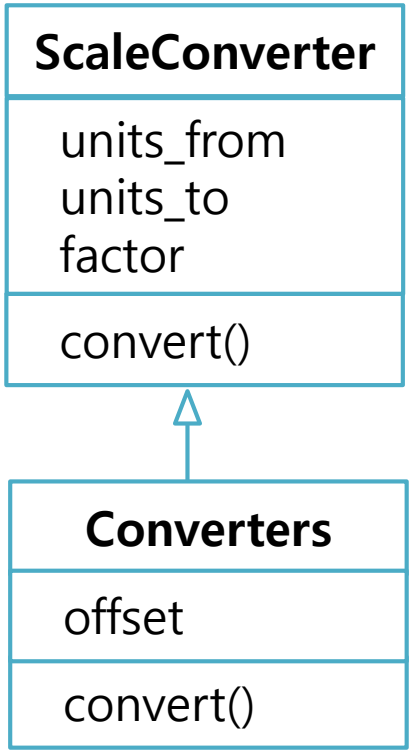
- 단위변환- inch(인치)를 mm로 변환하는 클래스

```
# inch를 mm로 변환하기 : 1inch -> 25mm
class ScaleConverter:
    def __init__(self, units_from, units_to, factor):
        self.units_from = units_from    # 단위
        self.units_to = units_to        # 변환할 단위
        self.factor = factor            # 변환 값

    def convert(self, value):  # 변환 계산 함수
        return self.factor * value  # 단위 기본값 x 수
```

```
if __name__ == "__main__":
    c = ScaleConverter("inches", "mm", 25)
    print("Converting 2 inches")
    print(str(c.convert(2)) + c.units_to)
```

상속 실습예제

- 단위 변환기 확장 클래스 만들기
  - 화씨온도(F) = 섭씨온도(C) x 1.8 + 32



상속 실습예제

- 단위 변환기 확장 클래스 만들기
  - 화씨온도(F) = 섭씨온도(C) x 1.8 + 32

```
from ch07.class_lib.scaleconverter import ScaleConverter
# 단위 변환 클래스 - 온도 변환
# F(화씨온도) = C(섭씨온도) * 1.8 + 32
class Converter(ScaleConverter):
    def __init__(self, units_from, units_to, factor, offset):
        super().__init__(units_from, units_to, factor)  #부모 클래스 멤버 상속
        self.offset = offset   # 단위 값

    def convert(self, value):
        return self.factor * value + self.offset  # (단위1 값 x 수) + 단위2 값

con = Converter('C', 'F', 1.8, 32)
print("Converting 20C")
print(str(con.convert(20)) + con.units_to)
```

실습 문제 - 클래스

아래의 프로그램을 분석하여 결과를 그 실행결과를 쓰시오

```
class City:
    a = ['Seoul', 'Incheon', 'Daejon', 'Jeju']

str = ''
for i in City.a:
    str += i[0]

print(str)
```