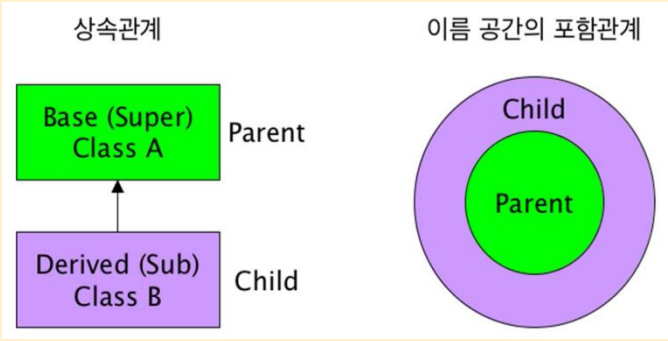
21.Python Class 상속

1.1 Class 상속

1] Class 상속 개념

- 상속받은 자식 클래스는 상속을 해준 부모 클래스의 모든 기능을 그대로 사용
- 자식 클래스는 필요한 기능만을 정의하거나 기존의 기능을 변경할 수 있음
- 파이썬에서는 클래스와 독립적으로 각 인스턴스를 하나의 이름 공간으로 취급



2] 생성자 호출

- 서브 클래스의 생성자는 슈퍼 클래스의 생성자를 자동으로 호출하지 않음
- 자식 클래스에 있는 init이 부모 클래스의 init을 overriding
- 서브 클래스의 생성자에서 슈퍼 클래스의 생성자를 명시적으로 호출
- 서브 클래스에 생성자가 정의되어 있지 않은 경우에는 슈퍼 클래스의 생성자가 호출

3] 메소드 Override

-서브 클래스에서 슈퍼 클래스에 정의된 메소드를 재정의하여 대치하는 기능

4] Polymorphism

- 상속 관계 내의 다른 클래스들의 인스턴스들이 같은 멤버 함수 호출에 대해 각각 다르게 반응
- 적은 코딩으로 다양한 객체들에게 유사한 작업을 수행(프로그램 작성 코드 량 감소 , 가독성 향상)

1.2 Class 상속 예시

예시 1

```
class Person:
    def __init__(self, name, phone=None):
        self.name = name
        self.phone = phone
    def __str__(self):
        return '<Person %s %s>' % (self.name, self.phone)

class Employee(Person):  # 괄호 안에 쓰여진 클래스는 슈퍼클래스를 의미한다.
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)  #   Person클래스의 생성자 호출
        self.position = position
        self.salary = salary

p1 = Person('서동범', 1498)
print ("p1.name->", p1.name)

m1 = Employee('이윤영', 5564, '대리', 200)
m2 = Employee('정재연', 8546, '과장', 300)
print ("m1.name, m1.position->", m1.name, m1.position)  # 슈퍼클래스와 서브클래스의 멤버를 하나씩 출력한다.
print ("m1->", m1)
print ("m2.name, m2.position->", m2.name, m2.position)
print ("m2->", m2     )
```

결과

```
------------------------------------------------------------
p1.name-> 서동범
p1-> <Person 서동범 1498>
m1.name, m1.position-> 이윤영 대리
m1-> <Person 이윤영 5564>
m2.name, m2.position-> 정재연 과장
m2-> <Person 정재연 8546>
```

1.3 class Polymorphism 예시

예시 1 (**상속 관계 내의 다른 클래스들의 인스턴스들이 같은 멤버 함수 호출에 대해 각각 다르게 반)**

```
 class Animal:        # Animal에 있는 cry를 Dog의 cry가 override 함
    def cry(self):
        print ('...')
class Dog(Animal):
    def cry(self):
        print ('멍멍')
class Duck(Animal):
    def cry(self):
        print ('꽥꽥')
class Fish(Animal):  # Fish는 cry가 없으므로 Animal의 내용을 그대로 사용
    pass

# in 뒤에는 튜플 안에 객체 3개 존재 -> each의 객체 형이 동적으로 결정되므로 그때마다의 cry가 다르게 사용
for each in (Dog(), Duck(), Fish()):
    each.cry()
```

결과

멍멍
꽥꽥
...

2.1 내장 자료형

1] 리스트 Sub Class

① 내장 자료형(list, dict, tuple, string)을 상속하여 사용자 클래스를 정의하는 것

② 내장 자료형과 사용자 자료형의 차이를 없에고 통일된 관점으로 모든 객체를 다룰 수 있는 방안

예시 1

```
class MyList(list):
    def __sub__(self, other): # '-' 연산자 중복 함수 정의
        for x in other:
            if x in self:
                self.remove(x) # 각 항목을 하나씩 삭제한다.
        return self

L = MyList([1, 2, 3, 'spam', 4, 5]) # L은 리스트이면서 1,2,3,'spam',4,5 원소를 가짐
print ("MyList([1, 2, 3, 'spam', 4, 5])->",L)      # print L에 대응되는 __str__이 없음

L = L - ['spam', 4]    # other → ['spam', 4]
print ("L - ['spam', 4]->", L)
```

결과

```
------------------------------------------------------------
MyList([1, 2, 3, 'spam', 4, 5])-> [1, 2, 3, 'spam', 4, 5]
L - ['spam', 4]-> [1, 2, 3, 5]
```

2.2 내장 자료형

예시 2 (**사전 서브 클래스 만들기)**

```
a = list()              # 상속을 받을 때 list를 상속 받음
print ("a->", a)        # 기본적으로 a는 일반적인 list
print (dir(a))
# 아래 예제는 keys() 메소드를  정렬된 키값  리스트를 반환하도록 재정의
class MyDict(dict):
    def keys(self):          # keys는 기존 dict에 존재하지만 재정의하여 사용
        K = dict.keys(self) # 언바운드 메소드 호출 --> K = self.keys() 라고 호출,무한 재귀 호출
        #  K.sort()
        return K

d = MyDict({'one':1, 'two':2, 'three':3})
print ("d.keys()->", d.keys())                  # dic.keys() → 클래스의 keys를 부르므로 언바운드 매소드

d2 = {'one':1, 'two':2, 'three':3}
print ("d2.keys()->", d2.keys())
```

결과

```
------------------------------------------------------------
a-> []
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
d.keys()-> dict_keys(['two', 'one', 'three'])
d2.keys()-> dict_keys(['two', 'one', 'three'])
```