상속

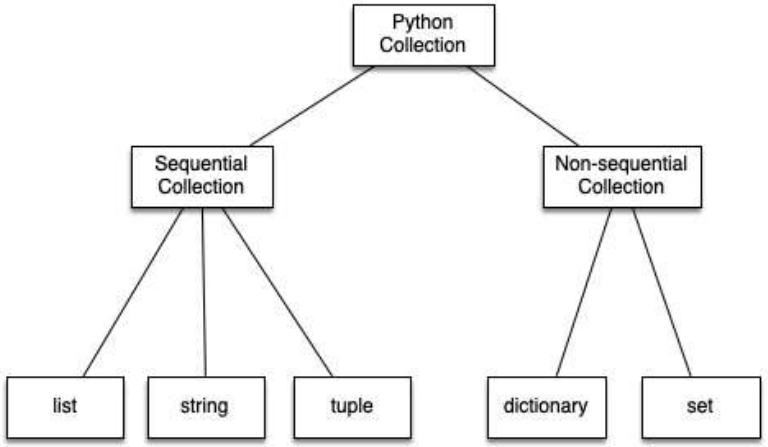
상속이란?

- **상속**inheritance: 객체 지향 프로그래밍의 또 다른 주요 요소
- 클래스를 선언할 때 다른 클래스의 속성과 메서드 상속 가능
- 상속을 받는 클래스: **자식 클래스** 또는 **하위 클래스**,
- 상속을 하는 클래스: **부모 클래스** 또는 **상위 클래스**

상속 활용 클래스 선언

```
class 자식클래스(부모클래스):
    클래스 본문
```

모음 자료형의 상속 체계


Python Collection
Sequential Collection
Non-sequential Collection
list
string
tuple
dictionary
set

Vector 클래스

- 벡터: 정수 또는 부동소수점으로 구성된 리스트 형식의 모음 자료형
- 길이가 동일한 두 벡터의 내적: 위치가 같은 두 항목의 곱셈의 합
- 내적 연산을 지원하도록 `list` 클래스의 기능 확장 필요

Vector 클래스 선언

```
class Vector(list):
    def __init__(self, items):
        super().__init__(items)
        self.len = self.__len__()

    # 벡터 내적
    def dot(self, other):
        if self.len != other.len:
            raise RuntimeError("두 벡터의 길이가 달라요!")
        sum = 0
        for i in range(self.len):
            sum += self[i] * other[i]
        return sum
```

```
>>> x = Vector([2, 3, 4])

>>> y = Vector([5, 6, 9])

>>> print(x) # __str__() 메서드 상속
[2, 3, 4]

>>> x.dot(y) # 내적: 2*5 + 3*6 + 4*9
64
```

메서드 재정의

- `append()` 메서드가 잘 작동하는 것처럼 보임
- 반면에 `len` 속성의 값이 4로 변하지 않음
- 그런데 `__len__()` 메서드와 `len()` 함수는 잘 작동

```
>>> x.append(5)

>>> x
[2, 3, 4, 5]

>>> x.len
3

>>> x.__len__()
4

>>> len(x)
4
```

append(), pop() 재정의

```
class Vector(list):

    ... (중략)

    # append() 메서드 재정의
    def append(self, item):
        super().append(item) # 부모 클래스의 append() 메서드 호출
        self.len = self.__len__()

    # pop() 메서드 재정의
    def pop(self, idx=-1):
        super().pop(idx)     # 부모 클래스의 pop() 메서드 호출
        self.len = self.__len__()
```

인덱싱, 슬라이싱

- 인덱싱과 슬라이싱은 관련 메서드를 재정의 하지 않았기에 리스트의 경우와 동일하게 작동

```
>>> x = Vector([2, 3, 4])

>>> x.append(5)

>>> x.pop()

>>> x.pop(1)

>>> x
[2, 4]

>>> x[0]
2

>>> x[:2]
[2, 4]
```

외부 함수 선언: 메서드 활용

- 아래 `dot()` 함수: 벡터의 내적을 함수로 지정하면 편리함
- `dot()` 함수를 호출하면 `Vector` 클래스의 `dot()` 메서드가 실행됨

```
def dot(x, y):
    assert isinstance(x, Vector) and isinstance(y, Vector)

    return x.dot(y)
```

```
>>> dot(x, y) == x.dot(y)
True
```

`__add__()` 매직 메서드 재정의: 벡터 합

```
class Vector(list):

    ... (중략)

    def __add__(self, other):
        # 벡터의 길이가 다르면 실행 오류 발생
        if self.len != other.len:
            raise RuntimeError("두 벡터의 길이가 달라요!")

        # 벡터 합 계산: 각 항목들의 합으로 이루어진 벡터
        new_list = []

        for i in range(self.len):
            item = self[i] + other[i]
            new_list.append(item)

        return Vector(new_list)
```

```
>>> x = Vector([2, 3, 4])

>>> y = Vector([5, 6, 9])

>>> x + y
[7, 9, 13]
```