31. 클래스

2018.12

일병 김재형

객체지향 프로그래밍

객체지향 프로그래밍의 시작

- ALGOL 60의 설계자인 Hoare가 1996년에 강연
  - 우리가 세계를 '사물"이라는 개념으로 이해한다.
  - '사물'은 테이블, 수식, 사람과 같은 것이다.
  - 우리의 사고, 언어, 행동은 사물을 설명하거나 조작하기위해 만들어졌다.
  - 따라서 현실 세계의 '사물' 모형을 컴퓨터내에 만들어야 한다.
- 사물: object(객체)
- 모형: model(모델)

변수-객체-review

객체(Object)

- 실생활에서 파악할 수 있는 것으로, 소프트웨어 세계에 구현할 대상
- 실행되는 프로그램에 존재하는 구조화된 데이터 덩어리
- 상태(state)와 행위(behavior)
  - 댐을 제어하는 시스템
  - 수문이 닫히고 열린 상태: 객체의 상태
  - 수문을 여는 행위(수문이 닫힌 상태에서 열린 상태로 변화): 행위

객체지향 프로그래밍

언어 설계자마다 다른 의미로 사용

- C++ 설계자: Bjarne Stroustrup
  - Class는 사용자 정의형을 만들기 위한 구조
  - 객체지향 프로그래밍이란 사용자 정의형과 상속을 사용한 프로그래밍
- Smalltalk 설계자: Alan Kay
  - 형에 대해서 반대하지는 않지만, 형과 상속은 좋지 않다.
  - 객체 지향이란 상태를 가진 객체가 메시지를 주고 받아서 커뮤니케이션하는 프로그램이다.

객체지향 프로그래밍

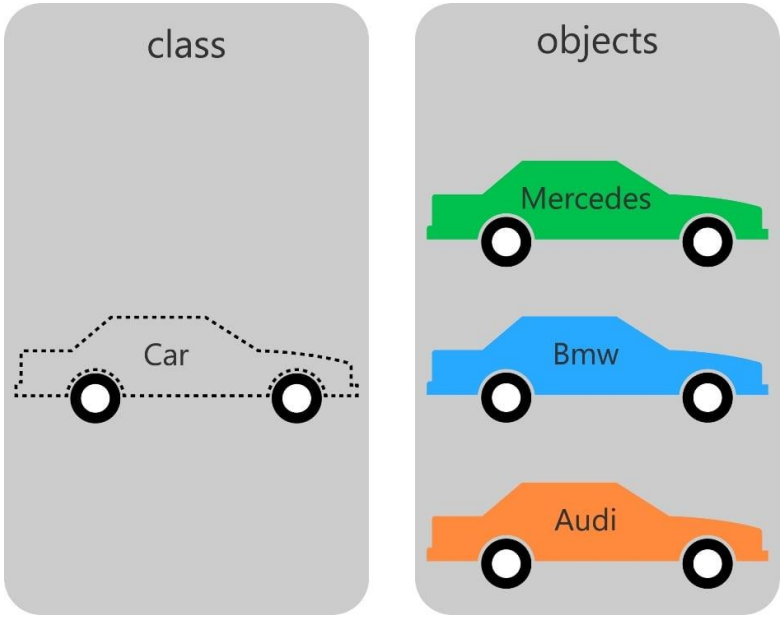
객체지향?

- 계수기(카운터): 물건의 개수를 샐 때 쓰는 물건
- 계수기의 상태
  - 지금까지 센 개수
  - 세는 물건의 이름
- 계수기의 행위
  - 개수를 증가
  - 개수를 초기화

객체지향 프로그래밍

객체지향?

- 함수와 변수: 하나의 정의에 하나의 컴퓨터 실체가 대응
- 현실: 비슷한 사물이 여러 개
- 같은 코드를 복사해서 사용하지 않는 방법이 필요



객체지향 프로그래밍

변수와 함수를 모아 모형을 만들자

- 모형을 만드는 여러가지 방법
  - 모듈(Module)
  - 함수와 변수를 동일하게 해쉬에 넣기
  - 클로저(Closure)
  - 클래스(Class)

변수-클래스-review

클래스(Class)

- 데이터를 추상화하는 단위
- 실생활의 사물을 소프트웨어로 구현하기 위해서는 추상화(Abstraction, 단순화하는 과정)가 필요.
- 같은 상태와 행위를 가진 객체는 같은 클래스이다.
- 속성(attribute)와 메서드(method)를 가진다.
  - 속성: 객체에 저장된 자료의 특성과 이름을 정의한 코드
  - 메서드: 객체의 행위를 구현한 함수(프로시저)

클래스(class)

초기(Hoare)의 클래스

- '현실 세계의 사물(object)은 편의상 상호 배타적 종류(classes)로 분류될 수 있다.'
- 처음 시작은 '분류'였다.

클래스(class)

C++의 클래스

- 클래스는 타입(type)이다.
- Int나 float같은 기본형처럼 다룰 수 있는 새로운 형을 사용자가 정의할 수 있게 하자.
- 사양으로서의 역할이 추가되었다.
  - 객체가 어떤 메소드를 가지고 있고 어떤 메소드를 가지고 있는가
  - 존재하지 않는 메소드를 호출하면 오류가 발생

클래스(class)

3가지의 역할

- 결합체를 만드는 생성기
  - 함수와 변수를 모아 하나의 객체를 생성
  - 클래스는 붕어빵을 만들기 위한 붕어빵 틀
- 어떤 조작이 가능한지 알려주는 사양
  - 기본형처럼 다루는 사용자 정의형
  - 메소드가 없으면 오류를 냄
  - 동적 형 결정 언어에서는 그다지 중요하게 여기지 않는다.
- 코드를 재사용하는 단위
  - 상속

클래스(class)

3가지의 역할

- C++이라고 해서 코드 재사용 단위의 역할이 없지는 않다.
- 언어마다 역할의 강약이 있다.

클래스(class)

항상 만들어야 하는가?

- 작은 규모일 경우에는 굳이 만들지 않아도 된다.
- 대규모일 경우에는 클래스를 사용해서 책임 범위를 나누는 것이 좋다.

클래스(class)

전체적인 그림

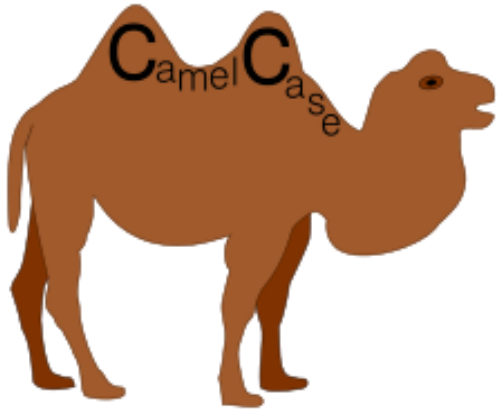
- 클래스
  - 클래스 변수(속성)
  - 클래스 메서드
- 인스턴스
  - 인스턴스 변수(속성)
  - 인스턴스 메서드

※ 클래스 안에 정의된 메소드는 클래스의 '맴버(Member)'라고 하기도 한다.

클래스(class)

PEP8

- 클래스 명은 카멜케이스(CamelCase)로 작성
  ex) Dog, Beagle, Person



변수-인스턴스-review

인스턴스(Instance)

- 인스턴스와 객체는 같은 의미이다.
- 하지만, 인스턴스는 '어떤 클래스에 속하는 특정 사례'라는 뜻으로 관계를 나타낸다.
- ex) '딸'=관계를 나타내는 단어
  '여자아이'=독립된 개념

클래스(class)

- 클래스
  - 클래스 속성과 메서드
    - 클래스 전체에 영향을 미치는 속성과 메서드
    - 클래스 및 인스턴스를 통해 호출
- 인스턴스
  - 인스턴스 속성과 메서드
    - 클래스에서 만들어진 각 인스턴스가 가진 속성과 그 인스턴스에만 영향을 미치는 메서드
    - 인스턴스를 통해 호출

클래스(class)-인스턴스

인스턴스 속성과 메서드 정의

- 계수기의 상태
  - 지금까지 센 개수
  - 세는 물건의 이름
- 계수기의 행위
  - 개수를 증가
  - 개수를 초기화

클래스(class)-인스턴스

인스턴스 속성과 메서드 정의

```
class 클래스명:
    def 인스턴스 메서드명(self, 매개변수):
        self.인스턴스 속성명 = 값
        수행문
```

```
인스턴스명 = 클래스명()
인스턴스명.속성 or 메서드()
```

클래스(class)-인스턴스

self?

- 관습적으로 사용되는 메서드의 첫 번째 매개변수 이름
- 메서드의 첫 번째 매개변수에 객체 자신을 넘긴다.
- 같은 클래스에서 생성한 여러 인스턴스 중 원하는 인스턴스에 있는 변수에 접근하기 위해서다.

클래스(class)-인스턴스

인스턴스명 = 클래스명()

- 클래스명()을 호출하면 인스턴스가 생성되어 인스턴스명에 할당

인스턴스명.속성 or 메서드()

- 인스턴스에 있는 속성(클래스, 인스턴스), 메서드에 접근하거나 호출한다

클래스(class)-인스턴스

isinstance(인스턴스, 클래스)

- 객체의 자료형을 판단시 사용

```
1  class Value:
2      pass
3
4
5  test = Value()
6  print(isinstance(test, Value))
7  num = 23
8  print(isinstance(num, int))
```

```
True
True
```

클래스(class)-인스턴스

계수기 만들기

```
1   class Counter:
2       def set_count(self, count=0):
3           self.count = count
4
5       def set_name(self, name):
6           self.name = name
7
8       def add(self, number=1):
9           self.counter += number
10
11      def reset(self):
12          self.counter = 0
13
14      def print_name(self):
15          print(self.name)
16
17      def print_count(self):
18          print(self.count)
```

```
21  sparrow_counter = Counter()
22  sparrow_counter.set_counter()
23  sparrow_counter.set_name("참새")
24  sparrow_counter.add(4)
25  sparrow_counter.print_name()
26  sparrow_counter.print_count()
27
```

```
참새
4
```

클래스(class)-인스턴스

오류

- 프로그래머가 깜빡함

```
21  sparrow_counter = Counter()
22  # sparrow_counter.set_count()
23  # sparrow_counter.set_name("참새")
24  sparrow_counter.add(4)
25  sparrow_counter.print_name()
26  sparrow_counter.print_count()
```

```
Traceback (most recent call last):
  File "Counter_Early.py", line 24, in <module>
    sparrow_counter.add(4)
  File "Counter_Early.py", line 9, in add
    self.count += number
AttributeError: 'Counter' object has no attribute 'count'
```

함수-특별한 이름-review

-와 __사용

- 두 언더스코어(__)로 시작하고 끝나는 함수
- 마술 메서드(maginc method), 특별 메서드라고 한다.
- 스티브홀던
  - 던더(double under->dunder) 메서드
- 파이썬에서 미리 정의된 함수로 이를 통해 사용자 객체를 만들 때, 기본적인 객체연산이 가능하도록 한다.
- ex) __getitem__()

클래스(class)-초기화

__init__

- initialize(초기화하다)를 줄여서 붙여짐
- 생성자(클래스를 인스턴스화 할 때, 호출되는 메서드, __new__)에 의해 인스턴스가 만들어진 후 호출된다.
- 이를 통해 인스턴스의 변수를 인스턴스화 할 때 명시적으로 초기화하게 할 수 있다.

클래스(class)-초기화

__init__

```
1   class Counter:
2       def __init__(self, name, count=0):
3           self.name = name
4           self.count = count
5
6       def add(self, number=1):
7           self.count += number
8
9       def reset(self):
10          self.count = 0
11
12      def print_name(self):
13          print(self.name)
14
15      def print_count(self):
16          print(self.count)
```

```
19  sparrow_counter = Counter("참새")
20  sparrow_counter.print_name()
21  sparrow_counter.print_count()
22  sparrow_counter.add()
23  sparrow_counter.print_count()
```

```
참새
0
1
```

클래스(class)-초기화

__init__

- 변수 할당을 잊어버리는 것 방지

```
20  sparrow_counter = Counter()
21  sparrow_counter.print_name()
22  sparrow_counter.print_count()
23  sparrow_counter.add()
24  sparrow_counter.print_count()
```

```
Traceback (most recent call last):
  File "Counter_init.py", line 20, in <module>
    sparrow_counter = Counter()
TypeError: __init__() missing 1 required positional argument: 'name'
```

클래스(class)-클래스

클래스속성과 메서드

- 클래스속성
  - 클래스라는 자료형의 정보가 정의되는 시점에 같은 메모리에 할당되는 속성
  - 클래스의 모든 인스턴스가 공유하는 속성
  - Ex) 총 계좌 개수
- 클래스 메서드
  - 클래스를 매개변수로 받는 메서드
  - 클래스와 인스턴스 양쪽에서 사용할 수 있다.

클래스(class)-클래스

클래스속성과 메서드 정의

```
class 클래스명:
    클래스 속성명 = 값

    @classmethod
    def 클래스 메서드(cls, 매개변수):
        수행문
```

※ cls는 class를 받는 변수로, 관례적으로 사용한다.

데커레이터

데커레이터(@)

- 사용자가 작성한 함수에, 미리 정의된 함수의 사용법을 추가하는 것
- @classmethod는 그 아래에 있는 메서드의 첫번째 매개변수에 인스턴스대신 클래스를 받도록 만든다
- 더 자세한 내용은 (심화)제너레이터, 이터레이터, 데커레이터에서 다룬다

클래스(class)-클래스

클래스속성과 메서드 정의

- 속성을 정의하고 인스턴스에서 값을 변경하면?

```
1  class Counter:
2      counter_num = 0
3
4      def __init__(self, name, count=0):
5          self.name = name
6          self.count = count
7
8      @classmethod
9      def print_class_num(cls):
10         print("계수기 클래스 속성 값: %d" % cls.counter_num)
11
12     def print_instance_num(self):
13         print("참새 계수기 인스턴스 속성 값: %d"
14               % self.counter_num)
15         self.counter_num += 1
```

```
17 sparrow_counter = Counter("참새")
18 Counter.print_class_num()
19 # 클래스 메서드는 인스턴스에서 사용가능
20 sparrow_counter.print_class_num()
21 print("참새 계수기 속성 값-클래스 직접 접근: %d"
22       % sparrow_counter.counter_num)
23 # 인스턴스 속성이 생성
24 sparrow_counter.print_instance_num()
25 Counter.print_class_num()
26 print("참새 계수기 인스턴스 속성 값: %d"
27       % sparrow_counter.counter_num)
```

```
계수기 클래스 속성 값: 0
계수기 클래스 속성 값: 0
참새 계수기 속성 값-클래스 직접 접근: 0
참새 계수기 인스턴스 속성 값: 0
계수기 클래스 속성 값: 0
참새 계수기 인스턴스 속성 값: 1
```

함수-네임스페이스와 스코프 -review

네임스페이스와 스코프

- 네임스페이스(Name space, 이름공간)
  파이썬내의 대응표를 이렇게 부른다.

- 변수의 스코프(scope, 범위)
  변수의 이름으로 그 변수가 가리키는 값을 찾을 수 있는 영역의 범위

클래스(class)-클래스

클래스/인스턴스 속성

- 스코프
  - 클래스가 인스턴스보다 상위에 있어 인스턴스가 없을 경우 클래스 속성을 사용
  - 인스턴스에서 속성에 값을 할당하면 인스턴스 속성이 된다.

클래스/인스턴스 메서드 사용

- 클래스
  - 클래스 메서드는 사용 가능하나, 인스턴스 메서드는 인스턴스를 받는 self에 넘겨줄 값이 없어 사용이 불가능하다.
- 인스턴스
  - 클래스 메서드, 인스턴스 메서드 둘 다 사용가능

```python
class Counter:
    counter_num = 0

    def __init__(self, name, count=0):
        self.name = name
        self.count = count

    @classmethod
    def print_class_num(cls):
        print("클래스 값: %d" % cls.counter_num)

    def print_instance_num(self):
        print("인스턴스 값: %d" % self.counter_num)
```

```python
# 직접 클래스 속성 접근 변경
Counter.print_class_num()
Counter.counter_num = 2
Counter.print_class_num()

# 인스턴스 변수 할당
sparrow_counter = Counter("참새")
sparrow_counter.counter_num = 5
print("인스턴스 값: %d" % sparrow_counter.counter_num)
print("클래스 값: %d" % Counter.counter_num)
sparrow_counter.print_instance_num()

# 클래스에서 인스턴스 메소드 실행
Counter.print_instance_num()
```

```
0
2
인스턴스 값: 5
클래스 값: 2
5
Traceback (most recent call last):
  File "Counter_class_instance.py", line 26, in <module>
    Counter.print_instance_num()
TypeError: print_instance_num() missing 1 required positional argument: 'self'
```

클래스(class)-클래스

클래스 변수와 메서드

- 계수기가 몇 개 만들어졌는지 기억한다.

```python
class Counter:
    counter_count = 0

    def __init__(self, name, count=0):
        self.name = name
        self.count = count
        self.add_counter()

    @classmethod
    def add_counter(cls):
        cls.counter_count += 1

    @classmethod
    def print_counter_count(cls):
        print("현재 계수기 개수: %d" % cls.counter_count)

    def add(self, number=1):
        self.count += number

    def reset(self):
        self.count = 0

    def print_name(self):
        print(self.name)

    def print_count(self):
        print(self.count)
```

```python
sparrow_counter = Counter("참새")
sparrow_counter.print_counter_count()
pigeon_counter = Counter("비둘기")
pigeon_counter.print_counter_count()
```

```
현재 계수기 개수: 1
현재 계수기 개수: 2
```

클래스(class)-클래스

클래스 변수와 메서드

- 인스턴스가 삭제될 때, 필요한 작업을 하고 싶다.

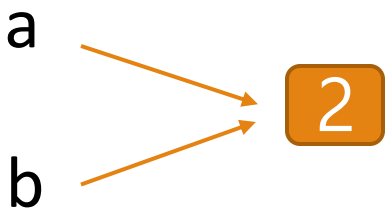
변수의 소멸-review

변수의 소멸

- del(변수)
- 변수를 제거하는 것

Garbage Collection(GC)

- 사용되지 않는 메모리 영역을 해제
- CPython에서는 레퍼런스 카운트 (reference count)를 사용.
- 레퍼런스 카운트(참조 수)가 0이 되면 메모리 영역을 해제



클래스(class)-finalizer

__del__

- 인스턴스가 파괴(메모리에서 해제)되기 직전에 호출
- 즉, 레퍼런스 카운트가 0이 될 때 호출
- Finalizer라 불리나 적당한 번역이 없다.

https://en.wikipedia.org/wiki/Finalizer

클래스(class)-finalizer

__del__

```
 9      def __del__(self):
10          self.del_counter()
11
12      @classmethod
13      def add_counter(cls):
14          cls.counter_count += 1
15
16      @classmethod
17      def del_counter(cls):
18          cls.counter_count -= 1
```

```
37  sparrow_counter = Counter("참새")
38  sparrow_counter.print_counter_count()
39  pigeon_counter = Counter("비둘기")
40  pigeon_counter.print_counter_count()
41  del(pigeon_counter)
42  Counter.print_counter_count()
```

```
현재 계수기 개수: 1
현재 계수기 개수: 2
현재 계수기 개수: 1
```

클래스(class)-상속

상속?

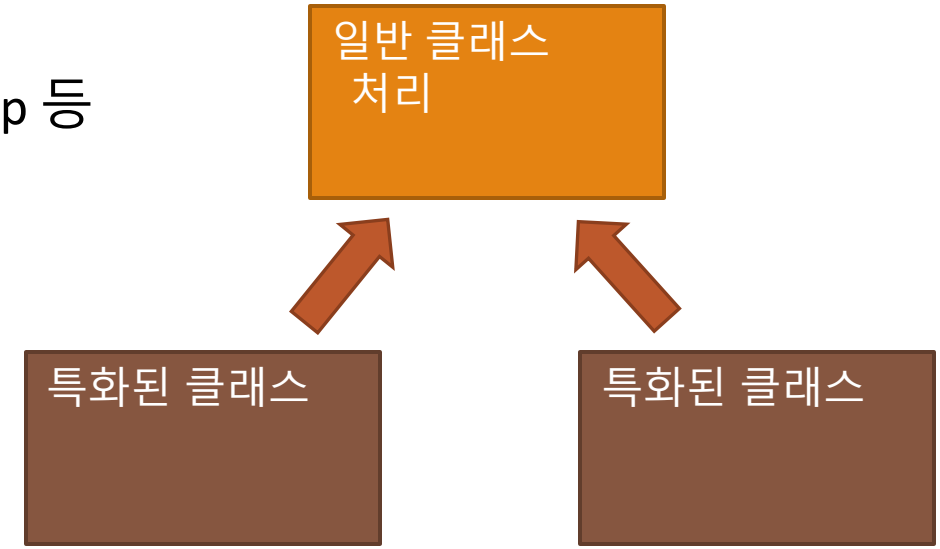
- Inheritance(상속)은 물려준다는 뜻이다.
- 클래스에서 상속은 속성과 메서드를 물려주는 것
- 부모클래스(기반 클래스(Base Class), 슈퍼 클래스)
  - 속성과 메서드를 주는 클래스
- 자식클래스(파생 클래스(Derived Class), 서브 클래스)
  - 부모클래스에서 속성과 메서드를 받는 클래스
- 동등한 관계일 때 사용(is-a 관계)
  - ex) 오리는 조류(오리 is a 조류)

클래스(class)-상속

상속에 관한 접근법(3가지)

- 일반화/특수화
  - 부모 클래스로 일반적인 기능을 구현
  - 자식 클래스로 목적에 특화된 기능 구현

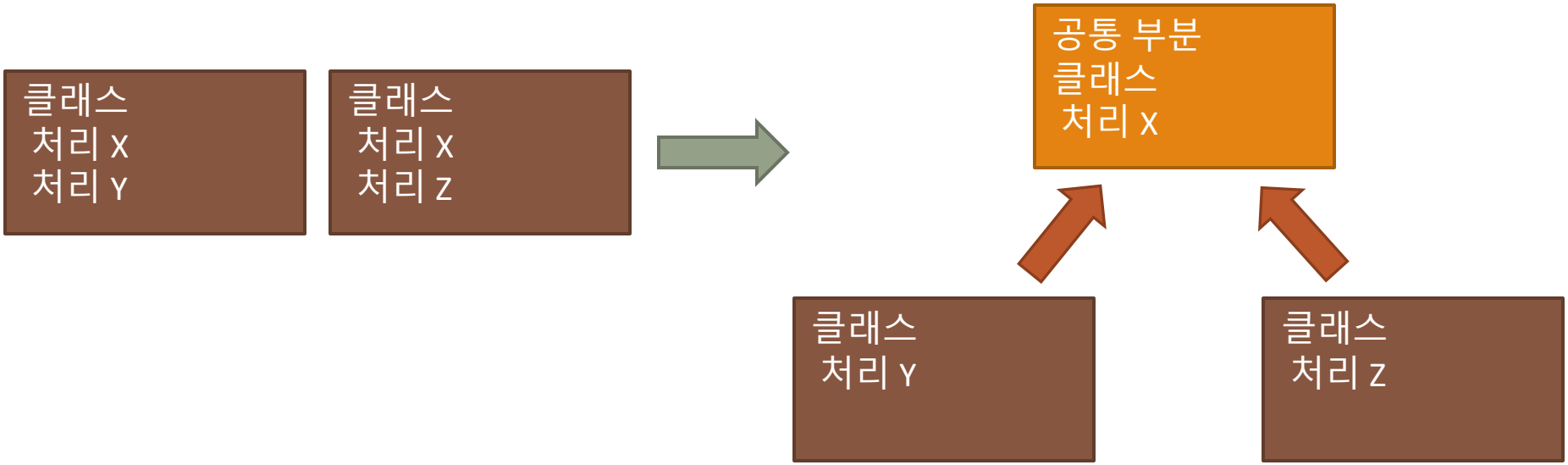
ex) 캐릭터 구현
부모 클래스: 이미지, 위치, hp 등
주 캐릭터: 사람이 조작
적 캐릭터: 컴퓨터가 조작



클래스(class)-상속

상속에 관한 접근법(3가지)

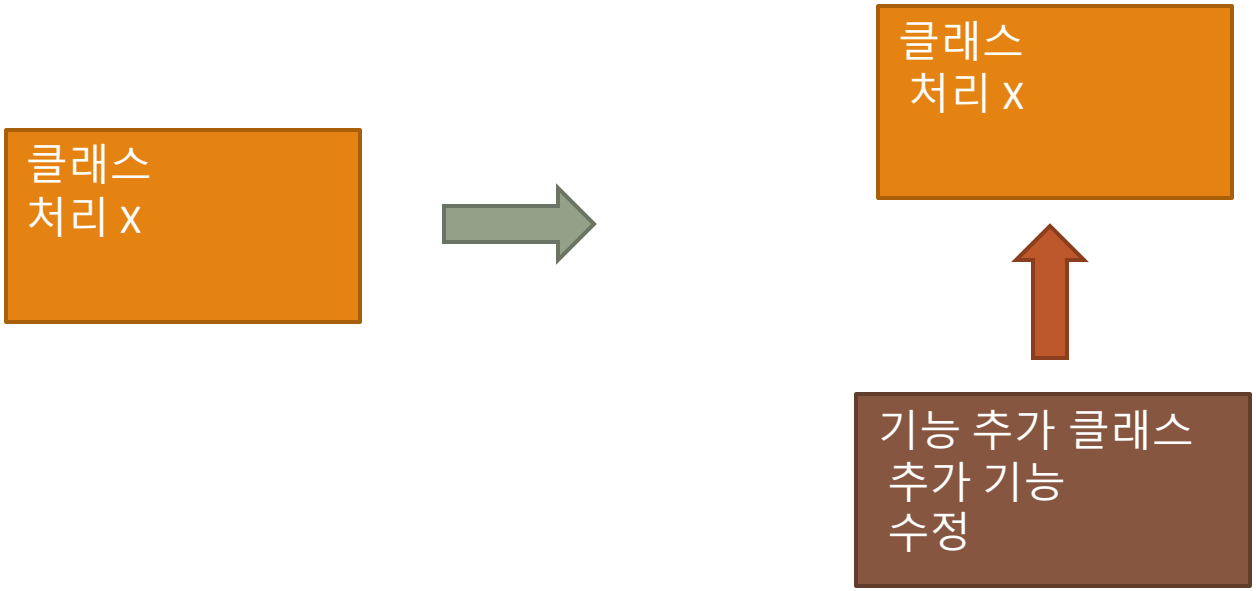
- 공통 부분을 추출
  - 복수 클래스의 공통 부분을 부모클래스로 추출



클래스(class)-상속

상속에 관한 접근법(3가지)

- 차분 구현
  - 상속 후 변경된 부분만을 구현하면 효율이 좋다.
  - 상속을 재사용을 위해 사용한다.



클래스(class)-상속

상속-양날의 칼

- 상속을 많이 사용하면 코드가 복잡하다.
- 깊은 상속 트리일 경우 객체가 메소드 x를 가질 때, 어떤 클래스에 있는지 알기 어려워진다.
- 코드 변경시, 영향 범위가 넓어져서 이해하기 어렵다.
- 특히, 차분 구현시 매우 깊어질 수 있다.

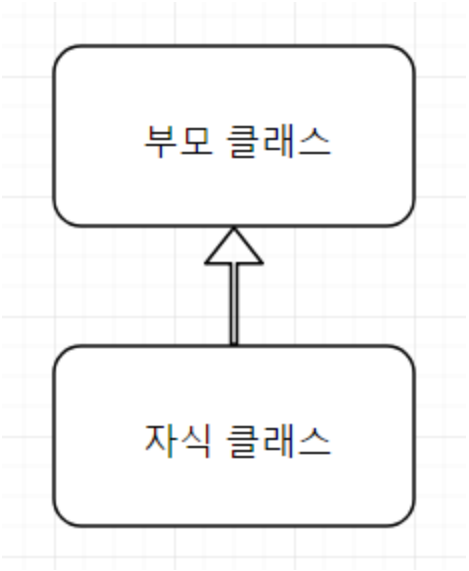
- 변수의 스코프를 정의한 이유와 유사하다.

클래스(class)-상속

상속 클래스 정의

```
class 부모클래스명:
    #내부 구현

class 자식클래스명(부모클래스명):
    #내부 구현
```



클래스(class)-상속

상속 클래스 정의

```
1   class Parent:
2       def parent_method(self):
3           print("부모 클래스 메서드")
4
5
6   class Child(Parent):
7       pass
8
9
10  parent = Parent()
11  parent.parent_method()
12
13  child = Child()
14  child.parent_method()
```

```
부모 클래스 메서드
부모 클래스 메서드
```

클래스(class)-상속

속성도 같이 물려받기

- 메서드 오버라이딩 때문에 이러한 문제가 발생

```
1   class Parent:
2       def __init__(self):
3           print("Parent 초기화")
4           self.attribute = "test"
5
6       def parent_method(self):
7           print("속성값: %s" % self.attribute)
8
9
10  class Child(Parent):
11      def __init__(self):
12          print("Child 초기화")
13
14
15  child = Child()
16  child.parent_method()
```

```
Child 초기화
Traceback (most recent call last):
  File "inheritance.py", line 16, in <module>
    child.parent_method()
  File "inheritance.py", line 7, in parent_method
    print("속성값: %s" % self.attribute)
AttributeError: 'Child' object has no attribute 'attribute'
```

클래스(class)-상속

속성도 같이 물려받기

- 파이썬 철학:
  "암묵적인 것보다는 명시적인 것이 좋다."
- 명시적으로 부모클래스의 __init__()를 호출해야한다.

```
1   class Parent:
2       def __init__(self):
3           print("Parent 초기화")
4           self.attribute = "test"
5
6       def parent_method(self):
7           print("속성값: %s" % self.attribute)
8
9
10  class Child(Parent):
11      def __init__(self):
12          Parent.__init__(self)
13          print("Child 초기화")
14
15
16  child = Child()
17  child.parent_method()
```

```
Parent 초기화
Child 초기화
속성값: test
```

클래스(class)-상속

super()

- 부모 클래스의 이름을 변경하거나, 부모클래스를 변경할 때, 자식 클래스 내부를 전부 변경해야 한다.
- 이를 해결하기 위해 super()를 사용한다.
- 어떤 메소드여도 가능하다.

```
1   class Parent:
2       def __init__(self):
3           print("Parent 초기화")
4           self.attribute = "test"
5
6       def parent_method(self):
7           print("속성값: %s" % self.attribute)
8
9
10  class Child(Parent):
11      def __init__(self):
12          super().__init__()
13          print("Child 초기화")
14
15
16  child = Child()
17  child.parent_method()
```

```
Parent 초기화
Child 초기화
속성값: test
```

클래스(class)
-메서드 오버라이딩

Method overriding

- Override: 무효로 하다. 우선하다.
- 객체지향 프로그래밍
  부모 클래스에서 받은 메서드를 재정의한다.

```
1   class Character:
2       def __init__(self):
3           self.inteligence = 10
4           self.strength = 10
5
6       def attack(self):
7           print("공격력: %f" % (self.strength*0.5))
8
9
10  class Warrior(Character):
11      def __init__(self):
12          super().__init__()
13          self.additional_strength = 20
14
15      def attack(self):
16          print("추가 공격력: %f" % (self.additional_strength*0.5))
17
18
19  warrior = Warrior()
20  warrior.attack()
```

```
추가 공격력: 10.000000
```

클래스(class)
-메서드 오버라이딩

Method overriding

- super()를 통해 부모의 메서드를 가져올 수 있다.

```
1   class Character:
2       def __init__(self):
3           self.inteligence = 10
4           self.strength = 10
5
6       def attack(self):
7           print("공격력: %f" % (self.strength*0.5))
8
9
10  class Warrior(Character):
11      def __init__(self):
12          super().__init__()
13          self.additional_strength = 20
14
15      def attack(self):
16          super().attack()
17          print("추가 공격력: %f" % (self.additional_strength*0.5))
18
19
20  warrior = Warrior()
21  warrior.attack()
```

```
공격력: 5.000000
추가 공격력: 10.000000
```

클래스(class) -연산자 정의

사칙연산이나 비교연산을 정의

- 특별메서드를 사용
- 일부만 서술하였다.

특별메서드 명	사용 방식
__add__(self, other)	self + other
__sub__(self, other)	self - other
__mul__(self, other)	self * other
__truediv__(self, other)	self / other
__eq__(self, other)	self == other
__ne__(self, other)	self != other

클래스(class) -연산자 정의

사칙연산이나 비교연산을 정의

- 사용자 정의형에서도 사칙연산이나 비교가 가능

```
1   class Character:
2       def __init__(self, name):
3           self.name = name
4           self.hp = 20
5           self.attack = 10
6           self.defence = 5
7
8       def __eq__(self, other):
9           return self.name == other.name
10
11      def __sub__(self, other):
12          self.hp = self.hp + self.defence - other.attack
13          print("현재 %s의 hp는 %d입니다." % (self.name, self.hp))
```

```
16  jaehyeong = Character("재형")
17  youngchun = Character("영천")
18  jaehyeong2 = Character("재형")
19
20  print("재형과 영천은 같은가?")
21  print(jaehyeong == youngchun)
22  print("재형2는 재형의 부계정인가?")
23  print(jaehyeong == jaehyeong2)
24
25  jaehyeong - youngchun
```

```
재형과 영천은 같은가?
False
재형2는 재형의 부계정인가?
True
현재 재형의 hp는 15입니다.
```

클래스(class)-포함

has-a 관계

- 포함을 컴포지션(composition),
  어그리게이션(aggregation)이라고도 한다.
- 클래스끼리 비교할 때 속하지는 않지만, 가지는 경우
- 속성에 다른 클래스의 인스턴스를 넣어 관리
- ex) 오리-부리와 꼬리

클래스(class)-포함

has-a 관계 예시

- ex) 오리-부리와 꼬리

```
1   class Bill():
2       def __init__(self, color):
3           self.color = color
4
5
6   class Tail():
7       def __init__(self, length):
8           self.length = length
9
10
11  class Duck():
12      def __init__(self, biil, tail):
13          self.bill = bill
14          self.tail = tail
15
16      def status(self):
17          print('이 오리는 %s의 부리와 %dcm 길이의 꼬리를 가집니다.'
18                % (self.bill.color, self.tail.length))
```

```
21  tail = Tail(10)
22  bill = Bill('주황색')
23  duck = Duck(bill, tail)
24  duck.status()
```

```
이 오리는 주황색의 부리와 10cm 길이의 꼬리를 가집니다.
```

클래스(class)
-비공개 속성과 보호 속성

속성의 비공개와 보호

- 부모클래스에서 내부적으로 상태를 나타내는 속성이 존재
- 자식클래스에서 실수로 같은 이름을 재정의
- 상속받은 메서드가 이상하게 작동할 수 있다.

- 이를 방지하기 위해 속성을 비공개로 사용할 수 있다.

클래스(class)
-비공개 속성과 보호 속성

속성의 비공개와 보호

```
1   class Dog():
2       def __init__(self):
3           self.value = "happy"
4
5       def get(self):
6           return self.value
7
8
9   class Beagle(Dog):
10      def __init__(self):
11          super().__init__()
12          self.value = 5
13
14
15  my_dog = Beagle()
16  print(my_dog.get(), '값과', my_dog.value, "값은 달라야 합니다.")
```

```
5 값과 5 값은 달라야 합니다.
```

클래스(class)
-비공개 속성과 보호 속성

Privated(비공개)

- 속성명 앞에 __(언더바 두 개)를 붙이고 뒤에 언더바가 없거나 하나의 언더바로 끝나도록 만든다.

  ex) __name, __name_

- 자식 클래스에서 실수로 접근하기 어려워진다.

```
1   class Dog():
2       def __init__(self):
3           self.__value = "happy"
4
5       def get(self):
6           return self.__value
7
8
9   class Beagle(Dog):
10      def __init__(self):
11          super().__init__()
12          self.__value = 5
13
14      def get_value(self):
15          return self.__value
```

```
18  my_dog = Beagle()
19  print(my_dog.get(), '값과', my_dog.get_value(), "값은 달라야 합니다.")
```

```
happy 값과 5 값은 달라야 합니다.
```

클래스(class)
-비공개 속성과 보호 속성

name mangling(이름 장식)

- 비공개속성은 다음과 같이 변경된다.
- _클래스이름__변수명
  ex) Dog클래스의 __name변수
  => _Dog__name
- 이러한 기능을 이름 장식(name mangling)이라 한다.

클래스(class)
-비공개 속성과 보호 속성

name mangling(이름 장식)

- 비공개속성이라도 직접 값을 할당할 수 있다.

```
1   class Dog():
2       def __init__(self):
3           self.__value = "happy"
4
5       def get(self):
6           return self.__value
7
8
9   class Beagle(Dog):
10      def __init__(self):
11          super().__init__()
12          self.__value = 5
13
14      def get_value(self):
15          return self.__value
```

```
18  my_dog = Beagle()
19  # __value로 새로운 속성 설정
20  my_dog.__value = 10
21  print("my_dog value의 값", my_dog.get_value())
22  print(my_dog.__dict__)
23  my_dog._Dog__value = "revise"
24  print(my_dog.get())
```

```
my_dog.__value의 값: 5
{'_Dog__value': 'happy', '_Beagle__value': 5, '__value': 10}
revise
```

클래스(class)
-비공개 속성과 보호 속성

Protected(보호)

- 이름 앞에 _(언더바 하나)를 붙인다.
- 파이썬 인터프리터가 특별히 처리하는 것은 없다.
- 관례적으로 파이썬 프로그래머 사이에서 그런 속성에 접근하지 않는 것을 금기로 사용한다.
- 몇몇 IDE에서는 _가 붙어있을 경우 플로팅 창에 속성을 안 띄워주기도 한다.

※ 마찬가지로 보호 메소드일 경우 _(언더바)하나를 붙인다.

클래스(class)
-비공개 속성과 보호 속성

Python에서 비공개 속성과 보호 속성
- 안전장치임으로 바꾸려면 바꿀 수 있다.
- 파이썬:
  "우리 모두는 성인이라는 사실에 동의한다"
- 실수로 바꾸는 것을 막아줄 뿐,
  의도적으로 수정하는 것은 막을 수 없다.

클래스(class)
-비공개 속성과 보호 속성

Python에서 비공개 속성과 보호 속성

- 비공개 속성을 통해 접근을 강제로 제어하지 않고, 보호속성을 사용하는 것이 좋다.
- 상속시, 자식클래스가 속성을 사용하기 어렵기 때문이다.
- 보호속성을 사용할 때는 문서화하여 자식 클래스가 속성을 사용할 때의 지침을 제공한다.
- 직접 제어할 수 없는 자식 클래스와 이름이 충돌하지 않게 할 때만 비공개 속성을 사용한다.

클래스(class)-getter와 setter

객체지향 프로그래밍

- 프로그램을 짤 때, 객체가 직접 다른 객체에 접근해서 속성을 변경하는 것은 좋지 않다.
- 메서드를 통해 속성이 변경되는 것이 좋다.
- Ex) npc에게서 물건을 살 때
- Alan Kay
  - 객체 지향이란 상태를 가진 객체가 메시지를 주고 받아서 커뮤니케이션하는 프로그램
  - 자세한 내용은 (심화)객체지향 프로그래밍에서 한다.

클래스(class)-getter와 setter

getter(획득자), setter(설정자)

- 변수에 직접 접근하지 않고,
  값을 획득하거나 설정하기 위해 사용하는 메서드
- getter(획득자)
  변수의 인스턴스를 통해 변수의 값을 얻는 메서드
- setter(설정자)
  변수의 값을 설정하는 메서드
- Python에서는 getter와 setter를 설정하면,
  변수에 값을 설정하고 획득하는 것처럼 사용 가능

클래스(class)-getter와 setter

getter(획득자), setter(설정자)

- 사용할 변수명 = property(getter함수명, setter함수명)
- 실제 내부 변수명과 사용할 변수명의 이름은 다르게

```python
class Person():
    def __init__(self, new_name):
        self._name = new_name

    def get_name(self):
        print("획득자 실행")
        return self._name

    def set_name(self, new_name):
        print("설정자 실행")
        self._name = new_name

    name = property(get_name, set_name)
```

```python
who = Person("재형")
print(who.name)
who.name = '영천'
print(who.name)
```

```
획득자 실행
재형
설정자 실행
획득자 실행
영천
```

https://docs.python.org/3/library/functions.html#property

클래스(class)-getter와 setter

getter(획득자), setter(설정자)

- 데코레이터 사용
- Getter에 사용할 이름으로 함수를 만듦
- Setter에 사용할 이름. setter를 데코레이터로 둔다.

```
1   class Person():
2       def __init__(self, new_name):
3           self._name = new_name
4
5       @property
6       def name(self):
7           print("획득자 실행")
8           return self._name
9
10      @name.setter
11      def name(self, new_name):
12          print("설정자 실행")
13          self._name = new_name
```

```
16  who = Person("재형")
17  print(who.name)
18  who.name = '영천'
19  print(who.name)
```

```
획득자 실행
재형
설정자 실행
획득자 실행
영천
```

클래스(class)-getter와 setter

getter(획득자)

- Setter가 없기 때문에 값을 설정할 수 없다.
- Property는 계산된 값을 참조하여 반환할 수 있다.

```
1   class Circle():
2       def __init__(self, radius):
3           self.radius = radius
4
5       @property
6       def diameter(self):
7           return 2 * self. radius
```

```
10  circle = Circle(10)
11  print("반지름")
12  print(circle.radius)
13  # 속성처럼 사용할 수 있다.
14  print("지름")
15  print(circle.diameter)
16
17  # 바로 계산이 된다.
18  circle.radius = 5
19  print("반지름이 5일 때 지름")
20  print(circle.diameter)
21
22  # 오류
23  circle.diameter = 20
```

```
반지름
10
지름
20
반지름이 5일 때 지름
10
Traceback (most recent call last):
  File "getter.py", line 23, in <module>
    circle.diameter = 20
AttributeError: can't set attribute
```

Exception class의 이해

Exception class의 구조

- Exception 클래스
  - 시스템 종료 외의 내장 예외는 모두 이 클래스에서 파생
  - 모든 사용자 정의 예외도 이 클래스를 상속
  - 이 클래스가 부모 클래스여서, 이를 통해 except를 받으면 다른 에러처리 방법이 무시되게 된다.

https://docs.python.org/ko/3/library/exceptions.html

Exception class의 이해

Exception class의 구조

```
BaseException
+-- SystemExit
+-- KeyboardIntterupt
+-- GeneratorExit
+-- Exception
      +-- …
      +-- ArithmeticError
            +-- ...
            +-- ZeroDivisionError
```

https://docs.python.org/ko/3/library/exceptions.html

예외 클래스 만들기

사용자 예외 클래스

- 단순하게 Exception클래스를 상속

```
1   class MyExceptionError(Exception):
2       pass
```

- 필요에 따라 속성이나 메서드를 추가해도 된다.
  - 초기화 메서드에 문자열을 넘기면, as 뒤의 에러메세지에 전달하는 값이 된다.

```
1   class MyExceptionError(Exception):
2       def __init__(self):
3           super().__init__("사용자 정의 예외 발생")
```

https://docs.python.org/ko/3/tutorial/errors.html

예외 클래스 만들기

PEP8

- 카멜케이스로 작성
- 실제 에러인 경우 Error를 뒤에 붙인다.

예외 클래스 만들기

정수 변환 오류 프로그램

```
1   class InvalidIntError(Exception):
2       def __init__(self, arg):
3           super().__init__("정수가 아닙니다.: %s" % arg)
4
5
6   def convert_int(text):
7       if text.isdigit():
8           return int(text)
9       else:
10          raise InvalidIntError(text)
11
12
13  try:
14      text = input("숫자를 입력하세요: ")
15      number = convert_int(text)
16  except InvalidIntError as e:
17      print('예외 발생 (%s)' % e)
18  else:
19      print("정수형식으로 변환하였습니다.")
```

```
숫자를 입력하세요: ad
예외 발생 (정수가 아닙니다.: ad)
```

```
숫자를 입력하세요: 123
정수형식으로 변환하였습니다.
```

상속 시 반드시 메서드 구현

NotImplementedError

- 구현되야 되는 부분이 구현되지 않았을 때 사용하는 오류

```
1   class Dog():
2       def bark(self):
3           raise NotImplementedError("bark메서드가 구현되지 않음")
4
5
6   class Beagle(Dog):
7       pass
8
9
10  my_dog = Beagle()
11  my_dog.bark()
```

```
Traceback (most recent call last):
  File "implemented_error.py", line 11, in <module>
    my_dog.bark()
  File "implemented_error.py", line 3, in bark
    raise NotImplementedError("bark메서드가 구현되지 않음")
NotImplementedError: bark메서드가 구현되지 않음
```

상속 시 반드시 메서드 구현

추상 기반 클래스(Abstract Base Classes)

- 자세한 내용은 (심화)Class에서 한다.

기본과제-자판기6

자판기(vending_machine.py)

- 배운 내용을 바탕으로 새로운 기능을 추가한다.
- 이전의 프로그램에 추가하시오.
- 물품 추가와 변경, 삭제 - admin모드
  - 1.물품을 추가하는 모드를 만든다.
    - 1)물품 추가시 물품명, 가격, 물품 개수를 입력받는다.
    - 2)추가된 물품은 맨 마지막에 추가된다.

기본과제-자판기6

자판기(vending_machine.py)

- 물품 추가와 변경, 삭제 - admin모드
  - 2.물품을 변경하는 모드를 만든다.
    - 1)어떤 물품을 변경할지 물어본다.
    - 2)물품명과 가격을 순차적으로 보여주면서 묻고, 값을 입력할 경우 입력된 값으로 변경한다.
    - 3)값을 입력하지 않고 엔터(빈 문자열)를 누를 경우, 원래 있었던 물품명이나 가격이 그대로 입력되게 한다.
  - 3.물품 삭제 모드를 만든다.
  - 1) 선택한 물품이 삭제된다.
  - 2) 물품이 삭제되었을 때 목록의 숫자가 비지 않도록 한다.

기본과제-자판기6

자판기(vending_machine.py)

- 오류 처리
  - 1. 파일 오류 시 오류 처리하기
  - 1) 파일이 없으면 파일을 생성하고, "관리자에게 연락하십시오."를 출력하고 종료 대신 기본 제어문(돈입력, 반환, 종료 등)을 출력한다.
  - 2) 파일내부에 오류가 있으면 "초기화할 때 오류가 있습니다. 관리자에게 연락하십시오."를 출력하고 자동 종료한다.
  - 2. 입력을 받는 모든 부분에서 원하는 입력 값이 아닐 경우 발생할 수 있는 오류를 처리한다.

기본과제-자판기6

자판기(vending_machine.py)

- 물품 추가와 변경, 삭제 예시

```
관리자모드
1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 1

추가할 물품을/를 입력하세요: python
추가할 가격을/를 입력하세요: 1000
추가할 개수을/를 입력하세요: 3

1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 4

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 3
3. 고급커피(200원) 개수: 3
4. python(1000원) 개수: 3
```

```
1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 2

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 3
3. 고급커피(200원) 개수: 3
4. python(1000원) 개수: 3
변경할 물품을 선택해주세요: 4
python을/를 선택하셨습니다.
물품 이름 변경(값 미입력시, 변경하지 않음):
물품 가격 변경(값 미입력시, 변경하지 않음): 10000

1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 4

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 3
3. 고급커피(200원) 개수: 3
4. python(10000원) 개수: 3
```

기본과제-자판기6

자판기(vending_machine.py)
- 물품 추가와 변경, 삭제 예시

```
1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 3

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 3
3. 고급커피(200원) 개수: 3
4. python(10000원) 개수: 3
삭제할 물품을 선택해주세요: 3
고급커피이/가 삭제되었습니다.

1. 물품추가
2. 물품변경
3. 물품삭제
4. 물품출력
5. 개수추가
6. 종료
원하는 작업을 선택해주세요: 4

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 3
3. python(10000원) 개수: 3
```

기본과제-자판기6

자판기(vending_machine.py)

- 파일이 없을 때 예외처리

```
-rwxrwxr--  1 root root 11135  2월  6 06:11 31.vending_machine.py
-rwxrwxr--  1 root root  7042  2월  5 02:48 README.md
-rwxrwxr--  1 root root    77  2월  6 06:13 item_list_.txt
root@goorm:/workspace/PythonSeminar18/Exercise/vending_machine(mas
관리자에게 연락하십시오.
1. 돈 입력
2. 거스름돈
3. 종료
현재까지 넣은 돈은 0원입니다.

뽑을 물품을 골라주세요:
```

기본과제-자판기6

자판기(vending_machine.py)

- 파일 내용에 오류가 있을 때 예외처리

```
root@goorm:/workspace/PythonSeminar18/
초기화할 때 오류가 있습니다.
관리자에게 연락하십시오.
root@goorm:/workspace/PythonSeminar18/
```

심화과제-LCD display

문제(lcd_display.py)

- 한 친구가 방금 새 컴퓨터를 샀다.
- 그 친구가 지금까지 샀던 가장 강력한 컴퓨터는 공학용 전자 계산기였다.
- 그런데 그 친구는 새 컴퓨터의 모니터보다 공학용 계산기에 있는 LCD 디스플레이가 더 좋다며 크게 실망하고 말았다.
- 그 친구를 만족시킬 수 있도록 숫자를 LCD 디스플레이 방식으로 출력하는 프로그램을 만들어보자.

http://codingdojang.com/scode/314

심화과제-LCD display

입력(lcd_display.py)

- 다음의 조건을 만족하는

  s n

  으로 입력한다.
- s는 숫자를 표시하는 크기(1 <= s < 10), n은 출력될 숫자(0<= n <= 99,999,999)를 의미한다.
- 0 이 두 개 입력되면 입력이 종료되고 프로그램을 끝낸다.

http://codingdojang.com/scode/314

심화과제-LCD display

출력(lcd_display.py)

- 수평 방향은 '-'기호, 수직 방향은 '|'를 이용해서 LCD 디스플레이 형태로 입력받은 숫자를 출력한다.
- 각 숫자는 정확하게 s+2개의 열, 2s+3개의 행으로 구성된다.
- 마지막 숫자를 포함한 모든 숫자를 이루는 공백은 스페이스로 채워야 한다.
- 두 개의 숫자 사이에는 정확하게 한 열의 공백(스페이스 1개)의 공백이 있어야 한다.

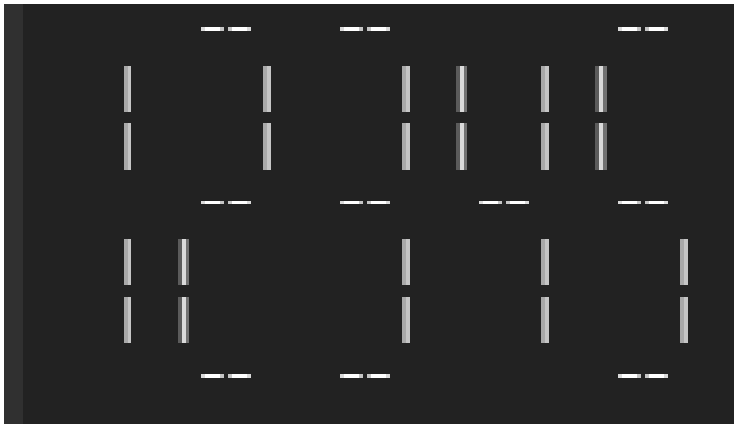
http://codingdojang.com/scode/314

심화과제-LCD display

예시

- 입력

2 12345



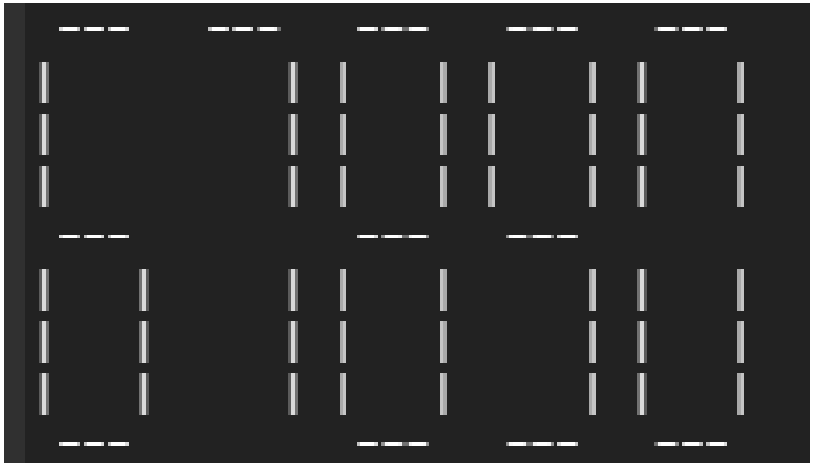
http://codingdojang.com/scode/314

심화과제-LCD display

예시

- 입력

3 67890



http://codingdojang.com/scode/314

심화과제-LCD display

예시

- 입력

0 0

출력 종료

http://codingdojang.com/scode/314