데이터 처리 프로그래밍

Data Processing Programming

08
객체와 클래스

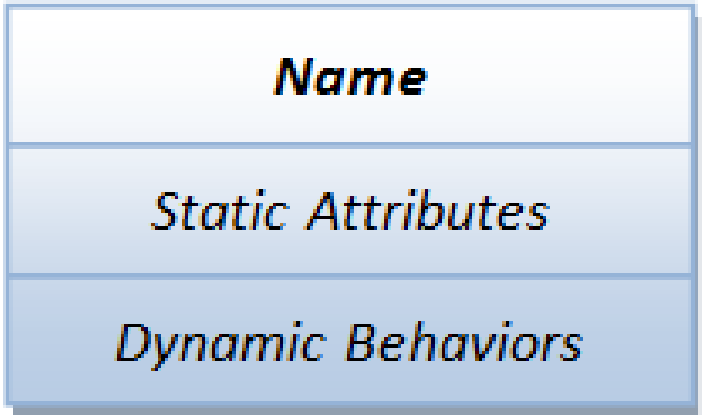
목차

1. 객체와 클래스
2. 인스턴스 변수, 클래스 변수
3. 클래스 매직 메소드
4. 클래스의 상속

1. 객체와 클래스

객체Object

- 객체란 존재하는 모든 것들을 의미
- 현실 세계는 객체로 이루어져 있고, 모든 사건들은 사물간의 상호작용을 통해 발생
- 객체란 객체의 속성을 이루는 데이터들 뿐만 아니라 그 데이터의 조작방법에 대한 내용도 포함
- 객체는 속성와 기능을 가지고 있는 것이 핵심

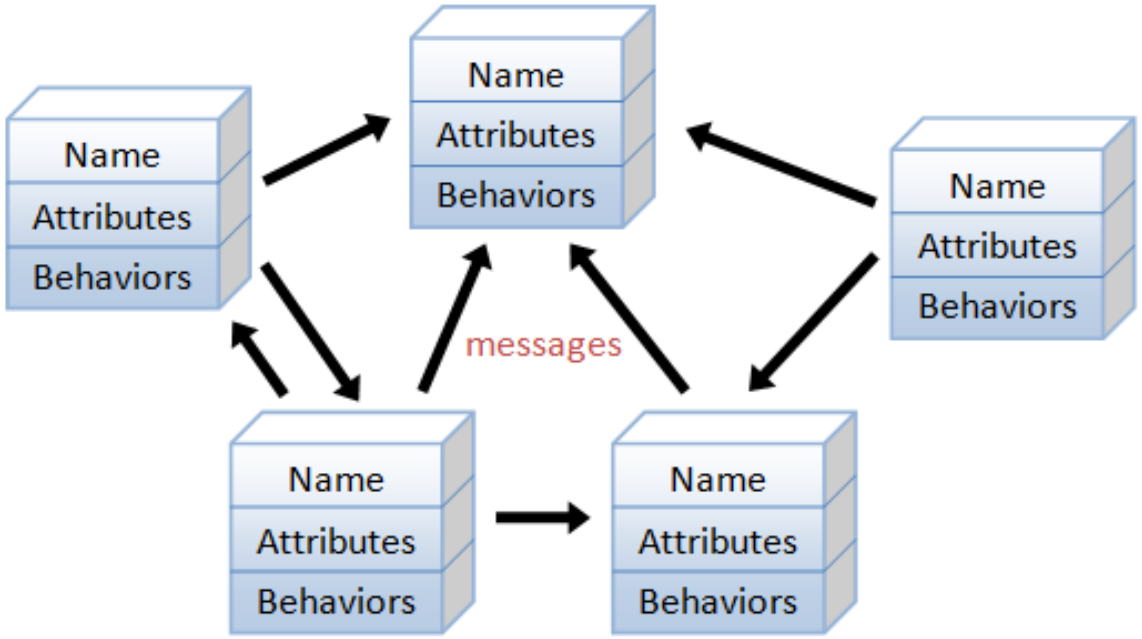


A class is a 3-compartment box

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/Python1a_OOP.html

객체 지향 프로그래밍Object Oriented Programming

- 객체 개념을 다루는 것이 객체 지향
- 객체 지향 프로그래밍은 컴퓨터 프로그래밍 기법 중 하나
- 프로그램을 단순히 데이터와 처리 방법으로 나누는 것이 아니라, 프로그램을 수많은 '객체'라는 단위로 구분하고, 이 객체들의 상호작용하는 방식
- 각각의 객체는 메시지를 주고 받고, 데이터를 처리



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/Python1a_OOP.html

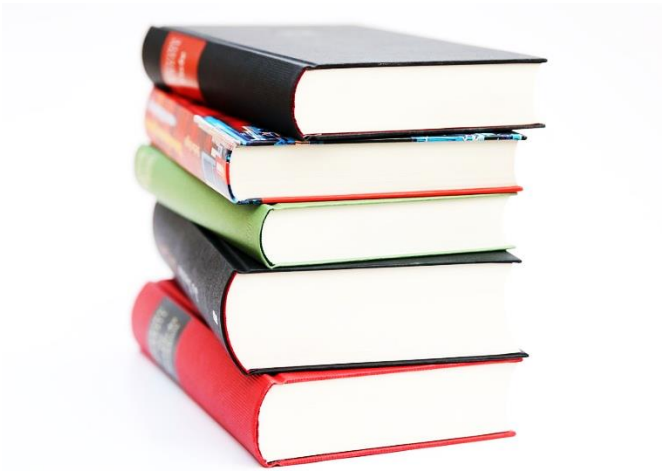
클래스Class

- 객체의 구성 요소를 담는 개념
- 여러 개의 변수(속성attribute)와 함수(메소드method) 들을 포함하는 개념
- 객체를 정의하는 틀 또는 설계도
- 실제 생성된 객체는 인스턴스Instance
- 인스턴스는 메모리에 할당된 객체를 의미

- 클래스 문법

```
class Book(object):
```

  - class: 클래스 예약어
  - Book: 클래스 명
  - object: 상속받는 객체명



Book 클래스 정의

- 클래스 이름: Book
- 속성
  - 저자: author
  - 책 이름: name
  - 출판사: publisher
  - 출판일: date

```
1   class Book(object):
2       author = ""
3       title = ""
4       publisher = ""
5       year = ""
6
7   b1 = Book()
8   b1.author = "Suan"
9   print(b1.author)
10  b1.title = "Data Programming"
11  print(b1.title)
```

```
Suan
Data Programming
```

Book 클래스의 메소드 정의

- 클래스 이름: Book
- 속성
  - 저자: author
  - 책 이름: name
  - 출판사: publisher
  - 출판일: date
- 메소드
  - 책 정보 출력: print_info(self)
  - self가 있어야만 실제로 인스턴스가 사용할 수 있는 메소드로 선언
  - print_info(self)에서 self는 실제적으로 b1 인스턴스를 의미
  - 메소드 안에서 속성 값을 사용하지 않을 경우에는 self 생략 가능

```
1   class Book(object):
2       author = ""
3       title = ""
4       publisher = ""
5       year = ""
6
7       def print_info(self):
8           print("Author: " + self.author)
9           print("Title: " + self.title)
10          print("Publisher: " + self.publisher)
11          print("Year: " + self.year)
12
13
14  b1 = Book()
15  b1.author = "Suan"
16  b1.title = "Data Programming"
17  b1.publisher = "KNU"
18  b1.year = "2019"
19  b1.print_info()
```

Lab. Book 클래스를 이용해 책 추가하기

- 자신만의 책 생성하기

```
1   class Book(object):
2       author = ""
3       title = ""
4       publisher = ""
5       year = ""
6
7       def print_info(self):
8           print("Author: " + self.author)
9           print("Title: " + self.title)
10          print("Publisher: " + self.publisher)
11          print("Year: " + self.year)
12
13
14
15
16
17
18
19
```

2. 인스턴스 속성, 클래스 속성

인스턴스 속성

- 인스턴스 속성은 객체로부터 인스턴스가 생성된 후에 인스턴스에서 활용하는 속성

클래스

저자	제목	출판사	발행일

인스턴스

저자	제목	출판사	발행일
저자	제목	출판사	발행일
저자	제목	출판사	발행일

인스턴스 변수

- Book 클래스에서 생성된 인스턴스 b1에서 변수를 활용

```
Author: Suan
Title: Data Programming
Publisher: KNU
Year: 2019
```

```python
class Book(object):
    author = ""
    title = ""
    publisher = ""
    year = ""

    def print_info(self):
        print("Author: " + self.author)
        print("Title: " + self.title)
        print("Publisher: " + self.publisher)
        print("Year: " + self.year)


b1 = Book()
b1.author = "Suan"
b1.title = "Data Programming"
b1.publisher = "KNU"
b1.year = "2019"
b1.print_info()
```

클래스 속성

- 클래스 속성은 클래스 자체에서 사용되는 속성

클래스

저자	제목	출판사	발행일

인스턴스

저자	제목	출판사	발행일
저자	제목	출판사	발행일
저자	제목	출판사	발행일

클래스 변수

- Book 클래스 자체에서 사용되는 변수

```
Author: Suan
Title: Data Programming
Publisher: KNU
Year: 2019
```

```
1   class Book(object):
2       author = ""
3       title = ""
4       publisher = ""
5       year = ""
6
7       def print_info(self):
8           print("Author: " + self.author)
9           print("Title: " + self.title)
10          print("Publisher: " + self.publisher)
11          print("Year: " + self.year)
12
13
14  b1 = Book()
15  Book.author = "Suan"
16  Book.title = "Data Programming"
17  Book.publisher = "KNU"
18  Book.year = "2019"
19  b1.print_info()
```

인스턴스 변수와 클래스 변수의 활용

- 인스턴스 변수
  - 저자 author
  - 제목 title
  - 출판사 publisher
  - 출판일 year
- 클래스 변수
  - 수량 count

클래스

저자	제목	출판사	출판일	수량

공유

인스턴스

저자	제목	출판사	출판일	수량
저자	제목	출판사	출판일	수량
저자	제목	출판사	출판일	수량

인스턴스 변수와 클래스 변수의 활용

- 인스턴스 변수와 클래스 변수를 목적에 맞도록 나누어서 활용

```
Author: Suan
Title: Data Programming
Publisher: KNU
Year: 2019
```

```python
class Book(object):
    author = ""
    title = ""
    publisher = ""
    year = ""
    count = 0

    def print_info(self):
        print("Author: " + self.author)
        print("Title: " + self.title)
        print("Publisher: " + self.publisher)
        print("Year: " + self.year)


b1 = Book()
Book.count += 1
b1.author = "Suan"
b1.title = "Data Programming"
b1.publisher = "KNU"
b1.year = "2019"
b1.print_info()
```

3. 클래스 매직 메소드

_ 기능

- for문을 사용하면서 iterator 역할을 위해서 i 변수가 필요
- for문 이후에는 iterator 변수 i가 필요하지 않음
- 이후에 사용되지 않을 변수에 어떤 이름을 부여하고 싶지 않을 때 _ 를 사용

```
>>> for i in range(5):
...     print("SuanLab")
...
SuanLab
SuanLab
SuanLab
SuanLab
SuanLab
```

```
>>> for _ in range(5):
...     print("SuanLab")
...
SuanLab
SuanLab
SuanLab
SuanLab
SuanLab
```

__ 기능

- _를 2개 붙여서 매직 메소드 또는 속성에 사용 가능
- 클래스에 포함된 매직 메소드
  - **__init__: 객체의 초기화를 위해 클래스 생성 시 호출되는 동작을 정의**
  - **__str__: 클래스의 인스턴스에서 str()이 호출될 때의 동작을 정의**
  - **__repr__: 클래스의 인스턴스에서 repr()이 호출될 때의 동작을 정의**
  - __new__: 객체의 인스턴스화에서 호출되는 첫 번째 메소드
  - **__del__: 객체가 소멸될 때 호출되는 메소드**
  - __dir__: 클래스의 인스턴스에서 dir()이 호출될 때의 동작을 정의
  - __getattr__: 존재하지 않는 속성에 엑세스하려고 시도할 때 행위를 정의
  - __setattr__: 캡슐화를 위한 방법 정의
  - **__add__: 두 인스턴스의 더하기가 일어날 때 실행되는 동작 정의**
  - **__lt__, __le__, __gt__, __ge__, __eq__, __ne__: 인스턴스 간의 <, <=, >, >=, ==, != 비교 메소드**
- __을 속성 앞에 붙이면 가시성을 위한 속성으로 사용

__init__() 메소드 이용

- __init__() 메소드를 이용하여 클래스의 속성들을 초기화

```
class Book(object):
    count = 0
    def __init__(self, author, title, publisher, year):
        self.author = author
        self.title = title
        self.publisher = publisher
        self.year = year
        Book.count += 1

    def print_info(self):
        print("Author: " + self.author)
        print("Title: " + self.title)
        print("Publisher: " + self.publisher)
        print("Year: " + self.year)


b1 = Book("Suan", "Data Programming", "KNU", "2019")
b1.print_info()
print(Book.count)
```

```
Author: Suan
Title: Data Programming
Publisher: KNU
Year: 2019
1
```

__str__() 메소드 이용

- __str__() 메소드를 이용하여 인스턴스 출력

```
Author: Suan
Title: Data Programming
Publisher: KNU
Year: 2019
1
```

```
1   class Book(object):
2       count = 0
3       def __init__(self, author, title, publisher, year):
4           self.author = author
5           self.title = title
6           self.publisher = publisher
7           self.year = year
8           Book.count += 1
9
10      def __str__(self):
11          return ("Author: " + self.author + \
12                  "\nTitle: " + self.title + \
13                  "\nPublisher: " + self.publisher + \
14                  "\nYear: " + self.year)
15
16
17  b1 = Book("Suan", "Data Programming", "KNU", "2019")
18  print(b1)
19  print(Book.count)
```

매직 메소드 예제: Line 클래스

```
1   class Line(object):
2       length = 0
3       def __init__(self, length):
4           self.length = length
5           print(self.length, '길이의 선 생성')
6
7       def __del__(self):
8           print(self.length, '길이의 선 삭제')
9
10      def __repr__(self):
11          return str(self.length)
12
13      def __add__(self, other):
14          return self.length + other.length
15
16      def __lt__(self, other):
17          return self.length < other.length
18
19      def __gt__(self, other):
20          return self.length > other.length
21
22      def __eq__(self, other):
23          return self.length == other.length
24
25      def __ne__(self, other):
26          return self.length != other.length
```

```
29  l1 = Line(10)
30  l2 = Line(20)
31  print(l1)
32  print(l2)
33  print('선의 합:', l1 + l2)
34  if l1 < l2:
35      print(l1, '<', l2)
36  elif l1 != l2:
37      print(l1, '!=', l2)
38  else:
39      pass
40
41  del(l1)
42  del(l2)
```

```
10 길이의 선 생성
20 길이의 선 생성
10
20
선의 합: 30
10 < 20
10 길이의 선 삭제
20 길이의 선 삭제
```

가시성 예제

- __items 속성은 Box 객체 외부에서 보이지 않도록 캡슐화와 정보 은닉이 가능
- 외부에서 __items 속성에 접근하면 속성 오류 발생

```
new item added
new item added
2
box1
```

```
Traceback (most recent call last):
  File "D:/Python/Box.py", line 18, in <module>
    print(b1.__items)
AttributeError: 'Box' object has no attribute '__items'
```

```
1   class Box(object):
2       def __init__(self, name):
3           self.name = name
4           self.__items = []
5
6       def add_new_item(self, item):
7           self.__items.append(item)
8           print("new item added")
9
10      def get_number_of_items(self):
11          return len(self.__items)
12
13  b1 = Box("box1")
14  b1.add_new_item("item1")
15  b1.add_new_item("item2")
16  print(b1.get_number_of_items())
17  print(b1.name)
18  print(b1.__items)
```

4. 클래스 상속

상속

- 클래스 상속Inheritance: 기존 클래스에 있는 속성과 메소드를 그대로 상속받아 새로운 클래스를 생성
- 공통된 클래스를 부모로 두고 자식들이 상속을 받아 클래스를 생성하므로 일관성 있는 프로그래밍 가능

- 클래스 상속

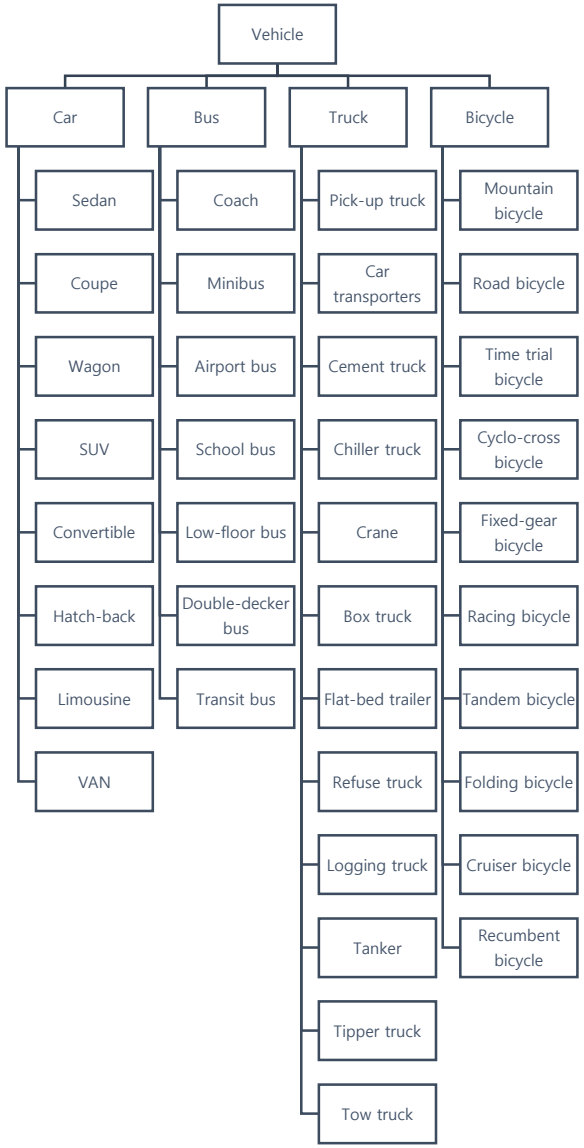
```
class SubClass(SuperClass):
```

클래스 상속, 추상 메소드, 메소드 오버라이딩

- SuperClass로부터 SubClass1와 SubClass2가 클래스 상속
- SuperClass의 method()는 정의되지 않은 추상 메소드
- SubClass1의 method()는 SuperClass의 추상 메소드를 오버라이딩

```
1  class SuperClass(object):
2      def method(self):
3          pass
4
5  class SubClass1(SuperClass):
6      def method(self):
7          print('Method overriding')
8
9  class SubClass2(SuperClass):
10     pass
11
12
13 sub1 = SubClass1()
14 sub2 = SubClass2()
15
16 sub1.method()
17 sub2.method()
```

상속 계층 예제

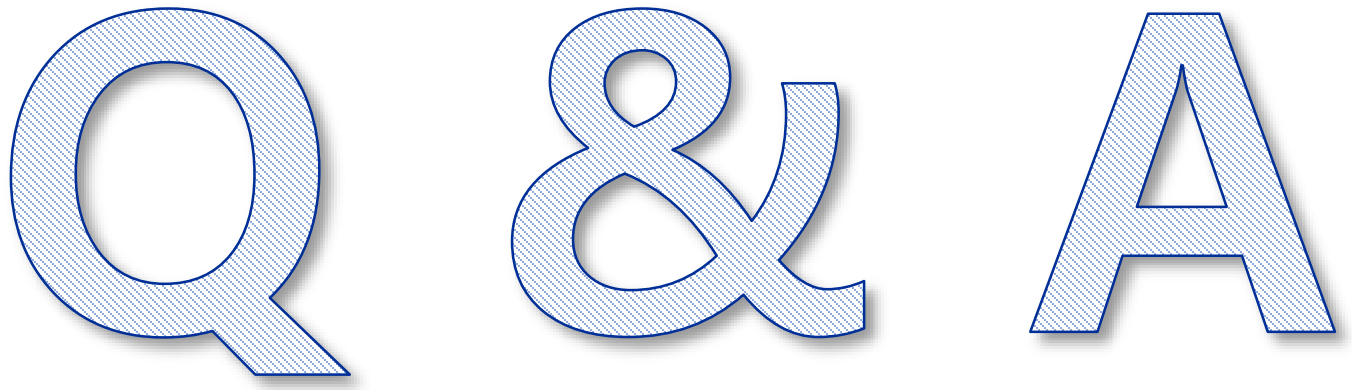

Vehicle
Car
Bus
Truck
Bicycle
Sedan
Coupe
Wagon
SUV
Convertible
Hatch-back
Limousine
VAN
Coach
Minibus
Airport bus
School bus
Low-floor bus
Double-decker bus
Transit bus
Pick-up truck
Car transporters
Cement truck
Chiller truck
Crane
Box truck
Flat-bed trailer
Refuse truck
Logging truck
Tanker
Tipper truck
Tow truck
Mountain bicycle
Road bicycle
Time trial bicycle
Cyclo-cross bicycle
Fixed-gear bicycle
Racing bicycle
Tandem bicycle
Folding bicycle
Cruiser bicycle
Recumbent bicycle

클래스 상속, 메소드 오버라이딩 예제

- Vehicle 클래스를 상속받아 Car 클래스와 Truck 클래스 생성
- Car 클래스와 Truck 클래스는 up_speed 메소드를 오버라이딩
- Car 클래스는 속도가 240 초과되면 240으로 조정
- Truck 클래스는 속도가 180 초과되면 180으로 조정

```
Speed: 240
Speed: 180
```

```
1  class Vehicle(object):
2      speed = 0
3      def up_speed(self, value): self.speed += value
4      def down_speed(self, value): self.speed -= value
5      def print_speed(self): print("Speed:", str(self.speed))
6
7  class Car(Vehicle):
8      def up_speed(self, value):
9          self.speed += value
10         if self.speed > 240: self.speed = 240
11
12 class Truck(Vehicle):
13     def up_speed(self, value):
14         self.speed += value
15         if self.speed > 180: self.speed = 180
16
17 c1 = Car()
18 t1 = Truck()
19
20 c1.up_speed(300)
21 c1.print_speed()
22 t1.up_speed(200)
23 t1.print_speed()
```


Q & A