객체와 클래스

객체(Object)

- 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 속성과 기능을 가지고 있는 것이 핵심

객체 지향 프로그래밍(Object 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

Name

클래스(Class)

- 객체의 구성 요소를 담는 개념
- 여러 개의 속성(Attribute)과 메소드(Method)를 포함하는 개념
- 객체를 정의하는 틀 또는 설계도
- 실제 생성된 객체는 인스턴스(Instance)
- 인스턴스는 메모리에 할당된 객체를 의미
- 클래스 문법

```
class Name(object):
```

  - class: 클래스 정의
  - Name: 클래스 명
  - object: 상속받는 객체명

Book 클래스 정의

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

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

```
book = Book()
book.author = "Suan"
print(book.author)
book.title = "Python Programming"
print(book.title)
```

```
Suan
```

```
Python Programming
```

Book 클래스 메소드 정의

- 메소드
  - 책 정보 출력: `print_info(self)`
  - `self`가 있어야만 실제로 인스턴스가 사용할 수 있는 메소드로 선언
  - `print_info(self)`에서 `self`는 실제적으로 `book` 인스턴스를 의미
  - 메소드 안에서 속성 값을 사용하지 않을 경우에는 `self` 생략 가능

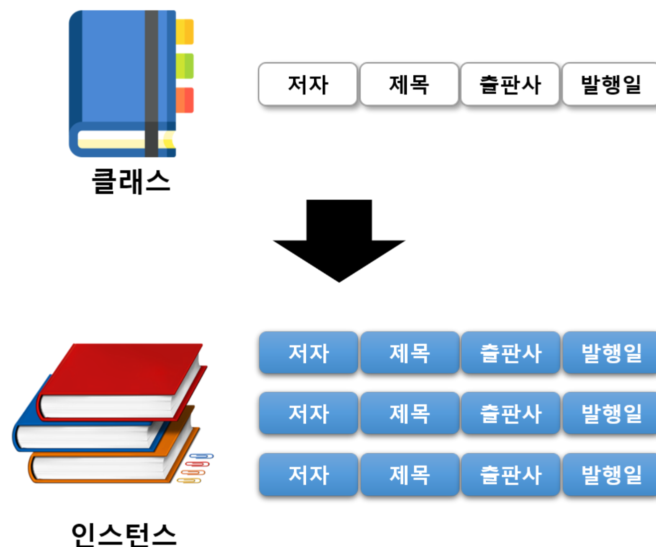
```
class Book(object):
  author = ""
  title = ""
  publisher = ""
  date = ""

  def print_info(self):
    print("Author:", self.author)
    print("Title:", self.title)
```

```
book = Book()
book.author = "Suan"
book.title = "Python Programming"
book.print_info()
```

```
Author: Suan
Title: Python Programming
```

인스턴스 속성(Instance Attribute)

- 인스턴스 속성은 객체로부터 인스턴스가 생성된 후에 인스턴스에서 활용하는 속성



Book 인스턴스 속성

- `Book` 클래스에서 생성된 인스턴스 `b1`에서 속성을 활용

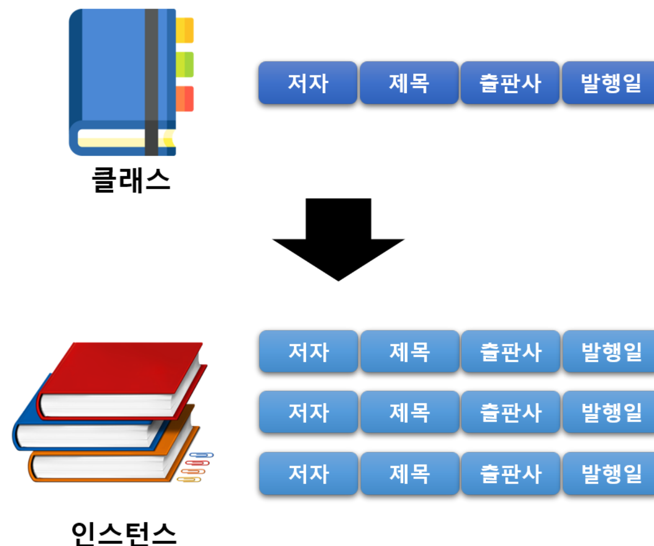
```
class Book(object):
  author = ""
  title = ""
  publisher = ""
  date = ""

  def print_info(self):
    print("Author:", self.author)
    print("Title:", self.title)
    print("Publisher:", self.publisher)
    print("Date:", self.date)
```

```
b1 = Book()
b1.author = "Suan"
b1.title = "Python Programming"
b1.publisher = "Colab"
b1.date = "2020"
b1.print_info()
```

```
Author: Suan
Title: Python Programming
Publisher: Colab
Date: 2020
```

클래스 속성(Class Attribute)

- 클래스 속성은 클래스 자체에서 사용되는 속성



Book 클래스 속성

- Book 클래스 자체에서 사용되는 속성

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

  def print_info(self):
    print("Author:", self.author)
    print("Title:", self.title)
    print("Publisher:", self.publisher)
    print("Date:", self.date)
```

```
b1 = Book()
Book.author = "Suan"
Book.title = "Python Programming"
Book.publisher = "Colab"
Book.date = "2020"
b1.print_info()
```

```
Author: Suan
Title: Python Programming
Publisher: Colab
Date: 2020
```

인스턴스 속성와 클래스 속성의 활용

- 인스턴스 속성과 클래스 속성을 목적에 맞도록 나누어서 활용

Book 인스턴스 속성과 클래스 속성

- 인스턴스 속성
  - 저자: author
  - 제목: title
  - 출판사: publisher

- 발행일: date
- 클래스 속성
    - 수량: count

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

  def print_info(self):
    print("Author:", self.author)
    print("Title:", self.title)
    print("Publisher:", self.publisher)
    print("Date:", self.date)
```

```
b1 = Book()
Book.count += 1
b1.author = "Suan"
b1.title = "Python Programming"
b1.publisher = "Colab"
b1.date = "2020"
b1.print_info()
print("Number of Books:", str(Book.count))
```

```
Author: Suan
Title: Python Programming
Publisher: Colab
Date: 2020
Number of Books: 1
```

클래스 매직 메소드(Class Masic Methods)

- '_'를 2개 붙여서 매직 메소드 또는 속성에 사용 가능
- __을 속성 앞에 붙이면 가시성을 위한 속성으로 사용
- 클래스 매직 메소드의 종류

매직 메소드	설명
`__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

매직 메소드	설명

`__init__()`

- `__init__()` 메소드를 이용하여 클래스의 속성들을 초기화

```
class Book(object):
  count = 0

  def __init__(self, author, title, publisher, date):
    self.author = author
    self.title = title
    self.publisher = publisher
    self.date = date
    Book.count += 1

  def print_info(self):
    print("Author:", self.author)
    print("Title:", self.title)
    print("Publisher:", self.publisher)
    print("Date:", self.date)
```

```
book = Book("Suan", "Python Programming", "Colab", "2020")
book.print_info()
print("Number of Books:", str(Book.count))
```

```
Author: Suan
Title: Python Programming
Publisher: Colab
Date: 2020
Number of Books: 3
```

`__str__()`

- `__str__()` 메소드를 이용하여 인스턴스 출력

```
class Book(object):
  count = 0

  def __init__(self, author, title, publisher, date):
    self.author = author
    self.title = title
    self.publisher = publisher
    self.date = date
    Book.count += 1

  def __str__(self):
    return ("Author:" + self.author + ₩
            "₩nTitle:" + self.title + ₩
            "₩nPublisher:" + self.publisher + ₩
            "₩nDate:" + self.date)
```

```
book = Book("Suan", "Python Programming", "Colab", "2020")
print(book)
print("Number of Books:", str(Book.count))
```

```
Author:Suan
Title:Python Programming
Publisher:Colab
Date:2020
Number of Books: 1
```

매직 메소드 예제

- Line 클래스

```
class Line(object):
  length = 0

  def __init__(self, length):
    self.length = length
    print(self.length, "길이의 선 생성")

  def __del__(self):
    print(self.length, "길이의 선 제거")

  def __repr__(self):
    return str(self.length)

  def __add__(self, other):
    return self.length + other.length

  def __lt__(self, other):
    return self.length < other.length

  def __le__(self, other):
    return self.length <= other.length

  def __gt__(self, other):
    return self.length > other.length

  def __ge__(self, other):
    return self.length >= other.length

  def __eq__(self, other):
    return self.length == other.length

  def __ne__(self, other):
    return self.length != other.length
```

```
l1 = Line(10)
print(l1)

l2 = Line(20)
print(l2)
```

```
print(l2)

print("선의 합:", l1 + l2)

if l1 < l2:
  print(l1, '<', l2)
elif l1 <= l2:
  print(l1, '<=', l2)
elif l1 > l2:
  print(l1, '>', l2)
elif l1 >= l2:
  print(l1, '>=', l2)
elif l1 == l2:
  print(l1, '==', l2)
elif l1 != l2:
  print(l1, '!=', l2)
else:
  pass

del(l1)
del(l2)
```

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

가시성 예제

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

```
class Box(object):
  def __init__(self, name):
    self.name = name
    self.__items = []

  def add_item(self, item):
    self.__items.append(item)
    print("아이템 추가")

  def get_number_of_items(self):
    return len(self.__items)
```

```
box = Box("Box")
box.add_item("Item1")
box.add_item("Item2")
print(box.name)
print(box.get_number_of_items())
#print(box.__items)
```

```
아이템 추가
아이템 추가
Box
2
```

클래스 상속(Class Inheritance)

- 기존 클래스에 있는 속성과 메소드를 그대로 상속받아 새로운 클래스를 생성
- 공통된 클래스를 부모로 두고 자식들이 상속을 받아 클래스를 생성하므로 일관성있는 프로그래밍 가능
- 기존 클래스에서 일부를 추가/변경한 새로운 클래스 생성으로 코드 재사용(reuse) 가능
- 클래스 상속 문법: `class SubClass(SuperClass):`

```
class SuperClass(object):
  pass

class SubClass(SuperClass):
  pass
```

메소드 오버라이딩(Method Overriding)

- `SuperClass` 로부터 `SubClass1` 와 `SubClass2` 가 클래스 상속
- 아무 내용도 없는 추상 메소드(Abstract Method) `method()` 를 정의
- `SubClass1` 의 `method()` 는 `SuperClass` 의 추상 메소드를 오버라이딩

```
class SuperClass(object):
  def method(self):
    pass

class SubClass1(SuperClass):
  def method(self):
    print("Method Overriding")

class SubClass2(SuperClass):
  pass
```

```
sub1 = SubClass1()
sub2 = SubClass2()

sub1.method()
sub2.method()
```

```
Method Overriding
```

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

```
class Vehicle(object):
  speed = 0
  def up_speed(self, value):
    self.speed += value

  def down_speed(self, value):
    self.speed -= value

  def print_speed(self):
    print("Speed:", str(self.speed))

class Car(Vehicle):
  def up_speed(self, value):
    self.speed += value
    if self.speed > 240: self.speed = 240

class Truck(Vehicle):
  def up_speed(self, value):
    self.speed += value
    if self.speed > 180: self.speed = 180
```

```
car = Car()
car.up_speed(300)
car.print_speed()

truck = Truck()
truck.up_speed(200)
truck.print_speed()
```

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