IT 개론

8장. 클래스와 객체지향 프로그래밍

목차

1. 데이터 추상화 개념

- 데이터가 컴퓨터 내에서 실제로 어떻게 표현되고 다루어지는가에 대한 정보는 숨기면서 데이터의 사용이 가능하도록 하는 방법이다.
- 데이터 추상화는 연관된 데이터들을 묶어서 하나의 데이터로 표현해야 하는 경우에 유용하다.

1. 데이터 추상화 개념

◆ 데이터 추상화 예

- 학생 데이터
  - (학번, 이름, 학과, 연락처)를 묶어서 하나의 학생 데이터로 표현할 수 있다.
- 지도상의 위치
  - (경도, 위도)를 묶어서 지도에서의 위치를 표현할 수 있다.
- 시간 데이터
  - (시, 분, 초)를 묶어서 시간을 표현할 수 있다.

2. 객체 지향 프로그래밍과 클래스 작성하기

- **OOP (Object-Oriented Programming)**
  - '객체지향 프로그래밍'이라고 한다.
  - OOP 방식을 이용하면 실생활의 물체들을 소프트웨어 객체로 표현하기 쉽다. 즉, 실생활에 가장 가까운 방식으로 데이터를 표현할 수 있도록 해 준다.
  - 실생활 객체와 같이, 소프트웨어 객체도 객체를 구성하는 데이터와 객체의 기능으로 표현한다.

객체(object) = 데이터 (data) + 기능 (functions)

attribute method

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 클래스 (class)

- 객체(object)를 만들기 위한 도구
- 클래스는 데이터를 표현하는 속성(attribute)과 데이터 기능을 표현하는 메소드(method)로 구성된다.
- 클래스의 구성

속성	멤버 데이터
메소드	멤버 데이터의 기능을 나타내는 함수
생성자, 소멸자	객체 생성과 소멸 시에 자동 호출되는 특별한 메소드
연산자 중복	연산자(+, - 등) 기호를 이용하여 표현할 수 있도록 함

2. 객체 지향 프로그래밍과 클래스 작성하기

```
class Person :
    name = 'Alice'
    age = 10
```

속성 (attribute)

```
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

생성자

```
    def __del__(self):
        pass
```

소멸자

```
    def ageUp(self, n):
        self.age += n
```

메소드 (method)

```
    def __add__(self, other):
        pass
```

연산자 중복

2. 객체 지향 프로그래밍과 클래스 작성하기

- **예제 1 - 강아지를 코드로 표현하기**
  - 강아지가 가지고 있는 속성 (attribute)
    - 이름
    - 기분
  - 강아지가 가지고 있는 기능 (method)
    - 멍멍짖기 (bark)
    - 꼬리흔들기

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 - 강아지를 코드로 표현하기

```
class Doggy:

    def talk(self):
        print("Hi. I am doggy.")
```

← Doggy 클래스 정의

- 클래스 Doggy는 talk 메소드만 하나 가지고 있는 클래스이다.
- 클래스 메소드는 함수의 형태이고 반드시 첫 번째 매개 변수로 self가 와야 한다.

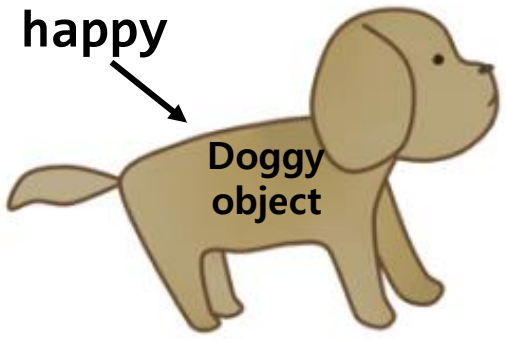
2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 - 강아지를 코드로 표현하기

```
class Doggy:

    def talk(self):
        print("Hi. I am doggy.")

# main

happy = Doggy()
happy.talk()
```

< 실행결과 >

```
>>>
Hi. I am doggy.
```



- 클래스 Doggy의 객체(object)를 만들려면 다음과 같이 해야 한다.

  happy = Doggy()

- talk() 메소드를 수행하려면 객체.메소드 형태로 실행시켜야 한다.

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 - 강아지를 코드로 표현하기

- Doggy 객체 생성하기

```
happy = Doggy()    # happy란 이름의 Doggy() 객체 생성함.
happy.talk()       # happy가 talk() 메소드를 호출함.
```

- Doggy 객체를 milky, choco라는 이름으로 두 개 더 만들기

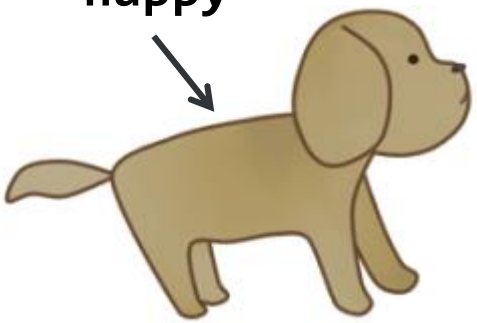
```
milky = Doggy()    # milky란 이름의 Doggy() 객체 생성함.
choco = Doggy()    # choco란 이름의 Doggy() 객체 생성함.
```

2. 객체 지향 프로그래밍과 클래스 작성하기

```
class Doggy:

    def talk(self):
        print("Hi. I am doggy.")
```

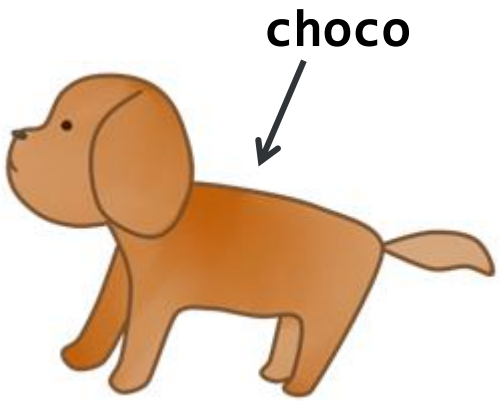
하나의 클래스에서 객체를 얼마든지 생성할 수 있다.

```
happy = Doggy()
milky = Doggy()
choco = Doggy()
```



happy.talk()

milky.talk()



choco.talk()

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 - 생성자 추가하기

- happy, milky, choco가 생성될 때 'Hello, I am born!' 이라는 메시지를 내고 싶다면 __init(self)__ 라는 이름의 특별한 메소드를 추가해야 한다.
- __init(self)__ - 생성자 (constructor)라고 부른다.
- 생성자는 객체 생성시에 자동으로 호출된다.

```
class Doggy:

  def __init__(self):
     print('Hello, I am born!')

  def talk(self):
     print("Hi. I am doggy.")
```

```
# main
happy = Doggy()
milky = Doggy()
choco = Doggy()
happy.talk()
milky.talk()
choco.talk()
```

```
>>>
Hello, I am born!
Hello, I am born!
Hello, I am born!
Hi. I am doggy.
Hi. I am doggy.
Hi. I am doggy.
```

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 – name 속성 추가하기

```
class Doggy:
    def __init__(self, name):
        self.name = name
        print('Hello, I am born!')

    def talk(self):
        print('Hi! I am', self.name)

# main
dog1 = Doggy('happy')
dog2 = Doggy('milky')
dog3 = Doggy('choco')

dog1.talk()
dog2.talk()
dog3.talk()
```

```
>>>
Hello, I am born!
Hello, I am born!
Hello, I am born!
Hi! I am happy
Hi! I am milky
Hi! I am choco
```

- Doggy 객체를 만들 때마다 각 객체는 자신의 속성 self.name을 갖게 된다.
- self.name 과 name은 다르다.
  - self.name 객체 속성.
  - name 매개변수.
  - name으로 전달받은 값이 생성되는 객체의 name 속성에 저장됨.

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 1 – mood 속성 추가하기

```
class Doggy:
  def __init__(self, name, mood):
     self.name = name
     self.mood = mood

  def talk(self):
     print('Hi! I am', self.name, 'and I am', self.mood, 'now')

# main
dog1 = Doggy('happy', 'HAPPY')
dog2 = Doggy('milky', 'SAD')
dog3 = Doggy('choco', 'BORING')

dog1.talk()
dog2.talk()
dog3.talk()
```

Doggy 객체 dog1, dog2, dog3는 각각 자기의 이름과 기분(mood)을 갖는다.

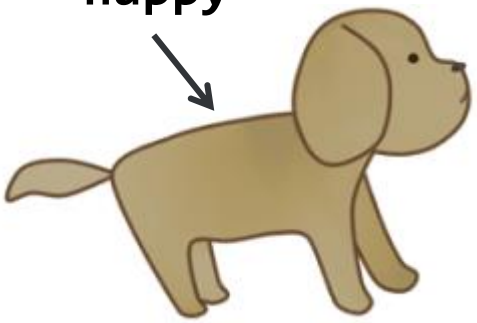
```
>>>
Hi! I am happy and I am HAPPY now
Hi! I am milky and I am SAD now
Hi! I am choco and I am BORING now
```

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ **예제 1** – 각 객체의 상태는 다음과 같다.

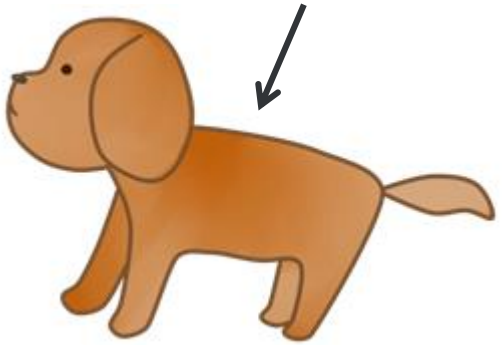
나의 이름은 happy
나의 기분은 HAPPY

happy



나의 이름은 choco
나의 기분은 BORING

choco



milky 나의 이름은 milky
나의 기분은 SAD

2. 객체 지향 프로그래밍과 클래스 작성하기

- 소멸자 - 객체가 소멸될 때 자동으로 호출되는 메소드

```
class Test:
        def __init__(self):
                self.value = 100
                print("created with value :", self.value)
        def __del__(self):  # 소멸자
                print("deleted with value :", self.value)
        def set(self, v):
                self.value = v
```

```
>>> x = Test()
created with value : 100

>>> x.set(200)
```

```
>>> x = Test()  # 새로 객체를 생성함.
created with value : 100
deleted with value : 200
```

새로 객체를 생성할 때 x 가 가리키던 이전 객체가 소멸된다. 이 때 자동으로 소멸자가 호출됨.

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ **예제 3** – 나 '서강이'는 학번이 20160000 이고 이번에 영어 시험에 95점을 받았어요.

나를 객체로 만들어 주세요!

< 속성 >

학번

이름

영어 성적

2. 객체 지향 프로그래밍과 클래스 작성하기

```
class Student:
    def __init__(self, no, name, score):
        self.no = no
        self.name = name
        self.score = score
    def printStudent(self):
        print("no    :", self.no)
        print("name  :", self.name)
        print("score :", self.score)

# main
s = Student(20160000, '서강이', 95)
s.printStudent()
```

```
no    : 20160000
name  : 서강이
score : 95
```

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 예제 4

```
import datetime

class Person :

    def __init__(self, name):
        self.name = name
        self.birthday = None

    def setBirthday(self, birthDate):
        self.birthday = birthDate

    def getAge(self):
        return (datetime.date.today() - self.birthday).days

    def getName(self):
        return self.name
```

2. 객체 지향 프로그래밍과 클래스 작성하기

```
me = Person('Taylor Swift')
him = Person('Barack Hussein Obama')
her = Person('Hillary Cliton')

me.setBirthday(datetime.date(1989, 12, 13))
him.setBirthday(datetime.date(1961, 8, 4))
her.setBirthday(datetime.date(1947, 10, 26))

personList = [me, him, her]
for p in personList:
    print(p.getName(), ':', p.getAge())
```

2. 객체 지향 프로그래밍과 클래스 작성하기

◆ 생성자와 소멸자

```
import time

class Life:
    def __init__(self):
        self.birth = time.ctime()
        print('Birthday', self.birth)

    def __del__(self):
        print('Deathday', time.ctime())

# main

life = Life()
del life
```

2. 객체 지향 프로그래밍과 클래스 작성하기

- 속성 (멤버데이터) 추가하기

```
class Person:
    name = "default"

p1 = Person()
p2 = Person()

print("p1's name : ", p1.name)
print("p2's name : ", p2.name)

p1.name = "Alice"
print("p1's name : ", p1.name)
print("p2's name : ", p2.name)
```

```
p1's name :  default
p2's name :  default
p1's name :  Alice
p2's name :  default
```

2. 객체 지향 프로그래밍과 클래스 작성하기

- 속성 (멤버데이터) 동적 추가하기

```
class Person:
    name = "default"

p1 = Person()
p2 = Person()
print(p1.name, p2.name)

Person.name = "Alice"
print(p1.name, p2.name)

p1.name = "Bob"
print(p1.name, p2.name)
```

```
default default
Alice Alice
Bob Alice
```

클래스

클래스변수와 인스턴스변수

```
class Account:
  num_accounts = 0     # 클래스변수

  def __init__(self, name):    # 생성자
       self.name = name        # 인스턴스변수
       Account.num_accounts += 1

  def __del__(self):           # 소멸자
       Account.num_accounts -= 1
```

클래스

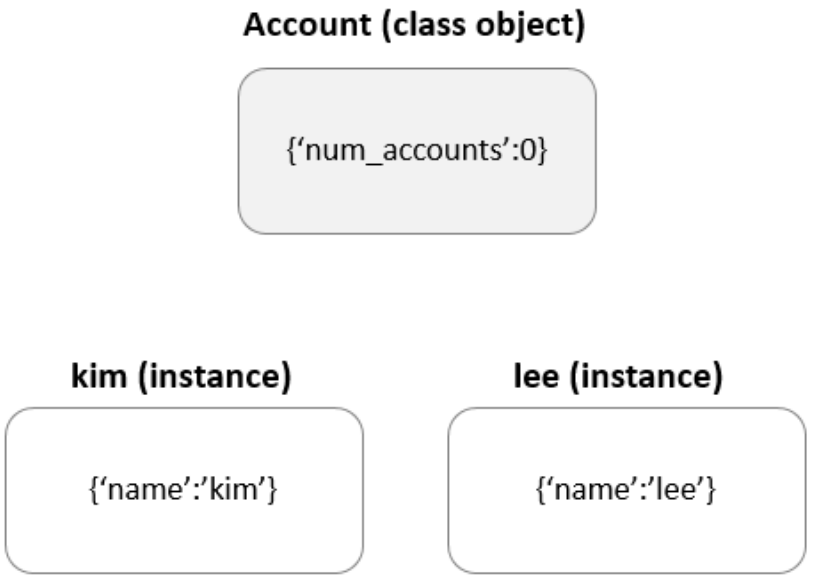
```
>>> kim = Account("kim")
>>> lee = Account("lee")

>>> kim.name
'kim'
>>> lee.name
'lee'

>>> kim.num_accounts
2
>>> lee.num_accounts
2
```

kim.num_accounts에서 먼저 인스턴스의 네임스페이스에서 num_accounts를 찾았지만 해당 이름이 없어서 클래스의 네임스페이스로 이동한 후 다시 해당 이름을 찾았고 그 값이 반환된 것

클래스

여러 인스턴스에서 공유해야 하는 값 => 클래스 변수 (클래스 네임스페이스)

```
>>> Account.num_accounts
2
>>> kim.num_accounts
2
>>> lee.num_accounts
2
```



클래스

상속

```
class Parent:
   def can_sing(self):
      print("Sing a song")

>>> father = Parent()
>>> father.can_sing()
Sing a song
```

```
>>> class LuckyChild(Parent):
pass

>>> child1 = LuckyChild()
>>> child1.can_sing()
Sing a song
```

클래스

```
>>> class UnLuckyChild:
pass


>>> child2 = UnLuckyChild()
>>> child2.can_sing()

Traceback (most recent call last): File "<pyshell#53>", line
1, in <module> child2.can_sing() AttributeError:
'UnLuckyChild' object has no attribute 'can_sing'
```

클래스

```
class LuckyChild2(Parent):
    def can_dance(self):
        print("Shuffle Dance")


>>> child2 = LuckyChild2()

>>> child2.can_sing()
Sing a song
>>> child2.can_dance()
Shuffle Dance
```