6. 객체지향

contents

1. 클래스와 객체

클래스와 객체

- 클래스(class)
  - 똑같은 무언가를 계속 만들어 낼 수 있는 설계 도면
- 객체(object)
  - 클래스로 만든 피조물

개념	설명	예시
객체(object)	실생활에 존재하는 실제적인 물건 또는 개념	심판, 선수, 팀
속성(attribute)	객체가 가지고 있는 변수	선수의 이름, 포지션, 소속팀
행동(action)	객체가 실제로 작동할 수 있는 함수, 메서드	공을 차다, 패스하다

1. 클래스와 객체

- **클래스와 객체**
  - 클래스로 만든 객체의 특징
    - 객체마다 고유한 성격을 가짐
    - 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않음
  - 파이썬 클래스의 가장 간단한 예

```
>>> class Cookie:
...     pass
```

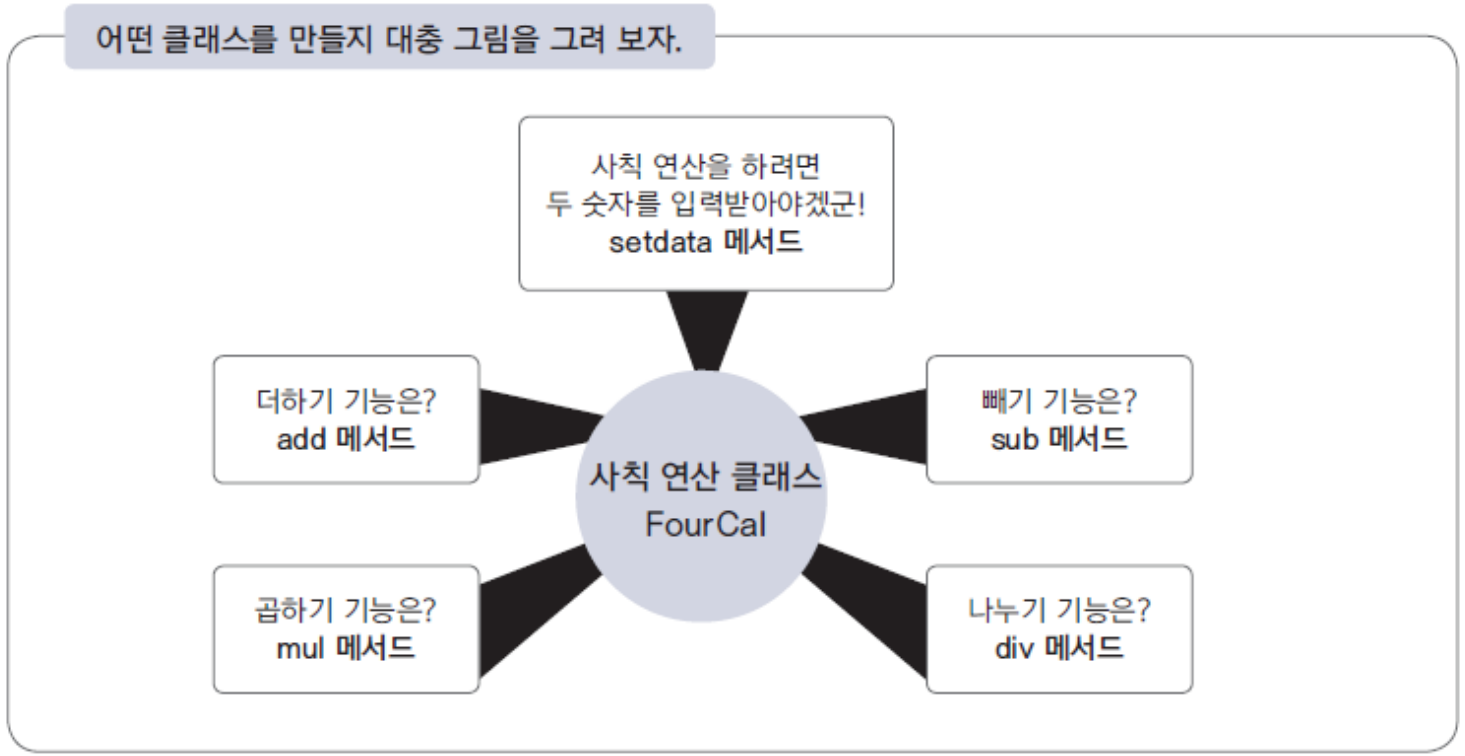
    - Cookie 클래스의 객체를 만드는 방법

```
>>> a = Cookie()
>>> b = Cookie()
```

1. 클래스와 객체

사칙연산 클래스 만들기

- 클래스를 어떻게 만들지 먼저 구상하기
  - 사칙연산을 가능하게 하는 FourCal 클래스 만들기



1. 클래스와 객체

사칙연산 클래스 만들기

클래스 구조 만들기

- Pass라는 문장만을 포함한 FourCal 클래스 만들기
- FourCal 클래스는 아무 변수나 함수도 포함하지 않지만 객체를 만들 수 있는 기능이 있음

```
>>> class FourCal:
...     pass
```

```
>>> a = FourCal()
>>> type(a)
<class '__main__.FourCal'> <— 객체 a의 타입은 FourCal 클래스이다.
```

1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

- 더하기 · 나누기 · 곱하기 · 빼기 등의 기능을 하는 객체 만들기
- 우선 객체에 사칙 연산을 할 때 사용할 2개의 숫자를 알려 주어야 함
- pass 문장을 삭제하고 setdata 함수 정의

```
>>> a.setdata(4, 2)
```

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
```

1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

- 메서드(method)
  - 클래스 안에 구현된 함수

- 일반적인 함수

```
def 함수_이름(매개변수):
    수행할_문장
    ...
```

- 메서드도 클래스에 포함되어 있다는 점만 제외하면 일반 함수와 다를 것이 없음

- 한 줄에 결괏값 출력하기

```
def setdata(self, first, second):  ← ① 메서드의 매개변수
    self.first = first
    self.second = second           ② 메서드의 수행문
```

1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

```
def setdata(self, first, second):
```
← ① 메서드의 매개변수

- setdata 메서드의 매개변수
  - self, first, second 3개의 입력값
  - 일반 함수와는 달리 메서드의 첫 번째 매개변수 self는 특별한 의미를 가짐
  - a 객체를 만들고 a 객체를 통해 setdata 메서드 호출하기

```
>>> a = FourCal()
>>> a.setdata(4, 2)
```

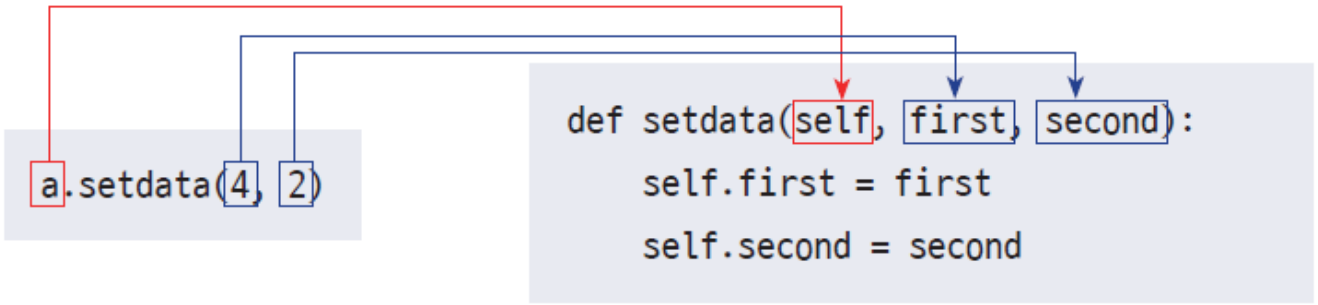
1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

```
def setdata(self, first, second):  <-- ① 메서드의 매개변수
```

- setdata 메서드의 매개변수
  - setdata 메서드에는 총 3개의 매개변수가 필요한데 실제로는 2개의 값만 전달하는 이유는?



  - setdata 메서드의 첫 번째 매개변수 self에는 setdata 메서드를 호출한 객체 a가 자동으로 전달되기 때문

1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

```
        self.first = first
        self.second = second
```

② 메서드의 수행문

- setdata 메서드의 수행문
  - a 객체에 객체변수 first와 second가 생성되고 지정된 값이 저장됨

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> a.first   <-- a 객체의 first 변숫값 출력
4
>>> a.second  <-- a 객체의 second 변숫값 출력
2
```

1. 클래스와 객체

사칙연산 클래스 만들기

객체에 연산할 숫자 지정하기

객체의 객체변수 특징 살펴보기

- a, b 객체 생성

```
>>> a = FourCal()
>>> b = FourCal()
```

- a 객체의 객체변수 first 생성

```
>>> a.setdata(4, 2)  <-- a 객체에 객체변수 first와 second가 생성되고 값 4와 2 대입
>>> a.first  <-- a 객체의 first 값 출력
4
```

- b 객체의 객체변수 first 생성

```
>>> b.setdata(3, 7)  <-- b 객체에 객체변수 first와 second가 생성되고 값 3과 7 대입
>>> b.first  <-- b 객체의 first 값 출력
3
```

1. 클래스와 객체

사칙연산 클래스 만들기

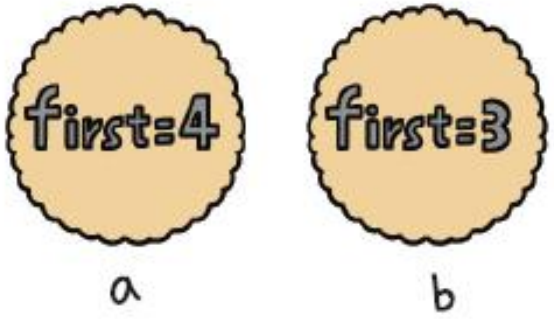
객체에 연산할 숫자 지정하기

객체의 객체변수 특징 살펴보기

- b 객체의 객체변수 first에 3이 저장됐을 때, a 객체의 first는 3으로 변할까? 아니면 기존 값 4를 유지할까?

```
>>> a.first
4
```

  - a 객체의 first 값은 b 객체의 first 값에 영향받지 않고 원래 값을 유지!



- 클래스로 만든 객체의 객체변수는 다른 객체의 객체변수와 상관없이 독립적인 값을 유지!

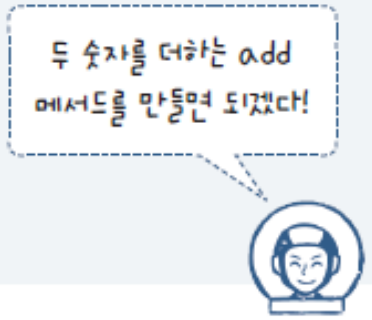
1. 클래스와 객체

사칙연산 클래스 만들기

더하기 기능 만들기

- 클래스에 2개의 숫자를 더하는 add 메서드 추가

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> a.add()
6
```




```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def add(self):
...         result = self.first + self.second
...         return result
```

- 클래스를 사용해 add 메서드 호출

```
>>> a = FourCal()
>>> a.setdata(4, 2)
```

```
>>> a.add()
6
```

1. 클래스와 객체

사칙연산 클래스 만들기

더하기 기능 만들기

add 메서드 자세히 살펴보기

```
def add(self):
    result = self.first + self.second
    return result
```

- add 메서드의 self에 객체 a가 자동으로 입력됨

```
result = self.first + self.second
```

- a.setdata(4, 2)가 먼저 호출되어
  a.first = 4, a.second = 2로 설정됨

- 결과

```
result = 4 + 2
```

```
>>> a.add()
6
```

1. 클래스와 객체

- **사칙연산 클래스 만들기**
  - 곱하기, 빼기, 나누기 기능 만들기
    - add 메서드와 동일한 방법으로 mul, sub, div 메서드 생성

```
...     def add(self):
...         result = self.first + self.second
...         return result
...     def mul(self):
...         result = self.first * self.second
...         return result
...     def sub(self):
...         result = self.first - self.second
...         return result
...     def div(self):
...         result = self.first / self.second
...         return result
```

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 8)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
```

```
>>> a.div()
2.0
>>> b.add()
11
>>> b.mul()
24
>>> b.sub()
-5
>>> b.div()
0.375
```

2. 생성자

- **생성자**
  - AttributeError 오류
    - FourCal 클래스의 객체 a에 setdata 메서드를 수행하지 않고 add 메서드를 수행하면, 'AttributeError: 'FourCal' object has no attribute 'first'' 오류 발생
    - setdata 메서드를 수행해야 객체 a의 객체변수 first와 second가 생성되기 때문

```
>>> a = FourCal()
>>> a.add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in add
AttributeError: 'FourCal' object has no attribute 'first'
```

2. 생성자

생성자

생성자(constructor)

- 객체가 생성될 때 자동으로 호출되는 메서드
- 메서드명으로 __init__ 사용

```
>>> class FourCal:
...     def __init__(self, first, second):
...         self.first = first
...         self.second = second
```

- 객체에 초깃값을 설정해야 할 필요가 있을 때 생성자를 구현하는 것이 안전한 방법
- 객체 생성 시 생성자의 매개변수에 해당하는 값을 전달해야 함

```
>>> a = FourCal(4, 2)
```

- __init__ 메서드의 매개변수에 전달되는 값

매개변수	값
self	생성되는 객체
first	4
second	2

- 결과 확인하기

```
>>> a = FourCal(4, 2)
>>> a.first
4
>>> a.second
2
```

실습1 : SoccerPlayer 클래스 정의

```
class SoccerPlayer(object):
    def __init__(self, name, position, back_number):
        self.name=name
        self.position=position
        self.back_number=back_number

    def change_back_number(self, new_number):
        print("선수의 등번호를 변경 : From %d to %d"%(self.back_number, new_number))
        self.back_number=new_number

    def __str__(self):
        return """
            My name is %s.
            I play in %s in center.
            My back_number is %d"""%(self.name, self.position, self.back_number)
```

실습1 : 객체 생성 및 등번호 변경

```
jinhyun=SoccerPlayer('Jinhyun',"MF", 10)
print(jinhyun)
jinhyun.change_back_number(5)
print(jinhyun)
```

실습 :2

Student

sno name gender major
__inin__() __str__() change_major()

① 학생 kim 을 학번(bno), 이름(name), 성별(gender), 전공(major) 초기 값을 설정하여 생성

② 생성된 학생 정보 출력

③ kim의 전공을 변경 후 학생정보 출력

실행결과

학번: 1, 이름 : 김수현, 성별 : 남자, 전공 : 컴공

전공을 컴공에서 정보통신으로 변경합니다.

학번: 1, 이름 : 김수현, 성별 : 남자, 전공 : 정보통신

3. 상속

클래스의 상속

- 상속(Inheritance)
  - '물려받다'라는 뜻
  - 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것

```
class 클래스_이름(상속할_클래스_이름)
```

```
class SubClass명(SuperClass명):
    …
```

  - FourCal 클래스를 상속하는 MoreFourCal 클래스

```
>>> class MoreFourCal(FourCal):
...     pass
```

3. 상속

- **클래스의 상속**
  - MoreFourCal 클래스
    - FourCal 클래스를 상속했으므로 FourCal 클래스의 모든 기능을 사용할 수 있어야 함

```
>>> a = MoreFourCal(4, 2)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2.0
```

3. 상속

- **클래스의 상속**
  - ab를 계산하는 MoreFourCal 클래스

```
>>> class MoreFourCal(FourCal):
...     def pow(self):
...         result = self.first ** self.second
...         return result
```

  - MoreFourCal 클래스로 만든 a 객체에 값 4와 2를 지정한 후 pow 메서드 호출

```
>>> a = MoreFourCal(4, 2)
>>> a.pow()
16
>>> a.add()
6
```

4. 메서드 오버라이딩

- **메서드 오버라이딩(method overriding)**
  - 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것
  - FourCal 클래스를 상속하는 SafeFourCal 클래스

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:  <— 나누는 값이 0인 경우, 값 0을 리턴하도록 수정
...             return 0
...         else:
...             return self.first / self.second
```

  - FourCal 클래스 대신 SafeFourCal 클래스를 사용

```
>>> a = SafeFourCal(4, 0)
>>> a.div()
0
```

4. 메서드 오버라이딩

- **다형성**
  - 상속에서 같은 이름의 메서드가 다른 기능을 할 수 있도록 하는 것

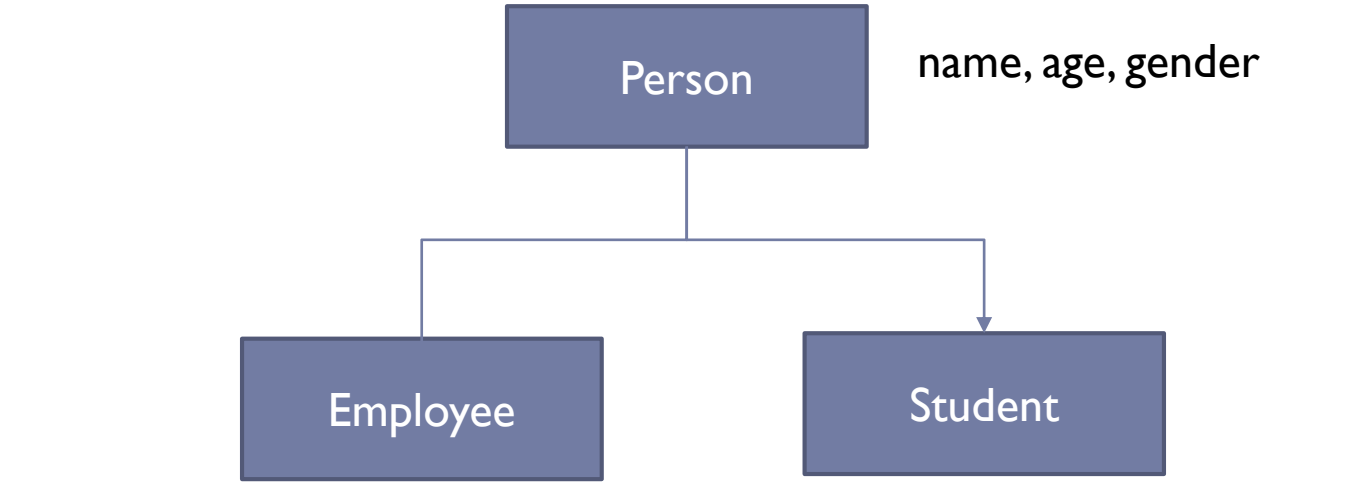
```
 1  class Animal:
 2      def __init__(self, name):
 3          self.name = name
 4      def talk(self):
 5          raise NotImplementedError("Subclass must implement abstract method")
 6
 7  class Cat(Animal):
 8      def talk(self):
 9          return 'Meow!'
10
11  class Dog(Animal):
12      def talk(self):
13          return 'Woof! Woof!'
14
15  animals = [Cat('Missy'), Cat('Mr. Mistoffelees'), Dog('Lassie')]
```

```
16  for animal in animals:
17      print(animal.name + ': ' + animal.talk())
```

```
Missy: Meow!
Mr. Mistoffelees: Meow!
Lassie: Woof! Woof!
```

↑ 15~17행 실행 결과

실습 3: 상속


Person
name, age, gender
Employee
Student
eno, name, age, gender, dept, sal
sno, name, age, gender, major

5. 클래스 변수

- **클래스변수**
  - 다른 객체들에 영향받지 않고 독립적으로 값을 유지하는 객체변수와는 다름
  - 클래스 안에 변수를 선언하여 생성
    - 예) Family 클레스에 클래스 변수 lastname 선언

```
>>> class Family:
...     lastname = "김"
```

```
>>> Family.lastname
김
```

5. 클래스 변수

- **클래스변수**
  - Family 클래스로 만든 객체를 통해서도 클래스 변수 사용 가능

```
>>> a = Family()
>>> b = Family()
>>> a.lastname
김
>>> b.lastname
김
```

- Family 클래스의 lastname을 변경하면?

```
>>> Family.lastname = "박"
>>> a.lastname
박
>>> b.lastname
박
```

  - 클래스로 만든 객체의 lastname 값도 모두 변경됨

6. 가시성

- **가시성**
  - 가시성visibility 은 객체의 정보를 볼 수 있는 레벨을 조절하여 객체의 정보 접근 제한 하는 것
  - 파이썬에서는 가시성이라고 하지만, 좀 더 중요한 핵심 개념은 캡슐화(encapsulation) 와 정보 은닉(information hiding)이다.]
  - 형식 : self.__변수명 : private 변수로 선언

```
class Person(object):
    def __init__(self, name, age, gender):
        self.__name=name
        self.age=age
        self.gender=gender

    def about_me(self):
        print("name:{}, age:{}, gender:{}".format(self.name, self.age, self.gender), end=" ")
```

0.0s

6. 가시성

- **가시성**
  - 가시성(private) 변수 외부에서 호출

```
print(p1.__name)
```

0.0s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[49], line 1
----> 1 print(p1.__name)

AttributeError: 'Person' object has no attribute '__name'
```

6. 가시성

- **가시성**
  - 가시성(private) 변수 외부에서 호출
    - @property 어노테션을 사용하여 외부에 사용가능하게 함

```
class Person(object):
    def __init__(self, name, age, gender):
        self.__name=name
        self.age=age
        self.gender=gender

    @property
    def getName(self):
        return self.__name
```

```
#p1.setName("park")
print(p1.getName)
```

✓ 0.0s

홍길동