프로그래밍

이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

클래스(Class)

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 계산기에 숫자 3을 입력하고 +를 입력한 후 4를 입력하면 결과값으로 7을 보여 준다. 다시 한번 +를 입력한 후 3을 입력하면 기존 결과값 7에 3을 더해 10을 보여 준다.
  - 힌트: 전역 변수(global)를 이용하여 이전에 계산한 결과값을 저장하여라

```
print(add(3))
print(add(4))
```

```
3
7
```

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라

```
# calculator.py
result = 0

def add(num):
    global result
    result += num  # 결괏값(result)에 입력값(num) 더하기
    return result  # 결괏값 리턴
```

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 만약 한 프로그램에서 2대의 계산기가 필요한 상황이 발생하면 어떻게 해야 할까?

```
# calculator2.py
result1 = 0
result2 = 0

def add1(num):  # 계산기1
    global result1
    result1 += num
    return result1

def add2(num):  # 계산기2
    global result2
    result2 += num
    return result2
```

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 계산기가 점점 더 많이 필요해 진다면 어떻게 해야 할까?
  - 계산기마다 빼기나 곱하기와 같은 기능을 추가해야 한다면 어떻게 해야 할까?

  - 위와 같은 경우에 클래스를 사용하면 다음과 같이 간단하게 해결할 수 있음

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 계산기가 점점 더 많이 필요해 진다면 어떻게 해야 할까?

```
# calculator3.py
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.add(3))
print(cal1.add(4))
print(cal2.add(3))
print(cal2.add(7))
```

클래스는 왜 필요한가?

- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 계산기가 점점 더 많이 필요해 진다면 어떻게 해야 할까?

```
# calculator3.py
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()
print(cal1.add(3))
print(cal1.add(4))
print(cal2.add(3))
print(cal2.add(7))
```

클래스

객체

클래스는 왜 필요한가?

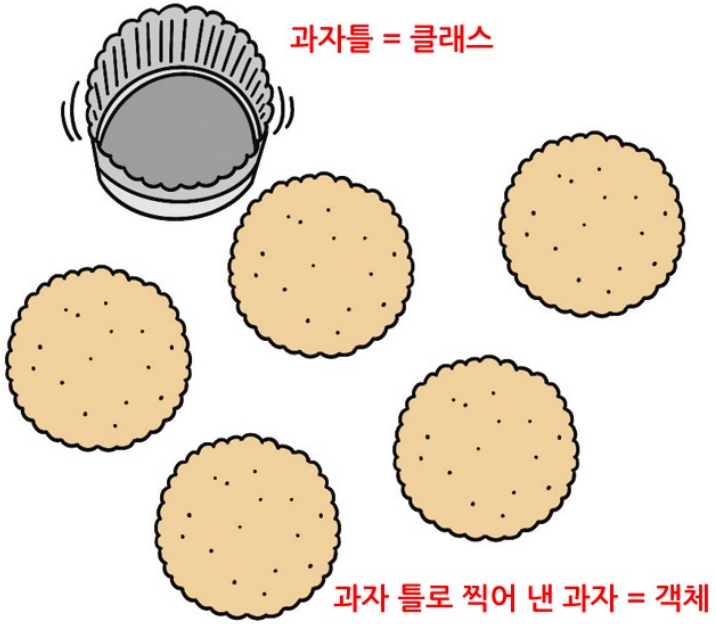
- '더하기' 기능을 가진 계산기를 파이썬 코드로 구현하라
  - 계산기마다 빼기나 곱하기와 같은 기능을 추가해야 한다면 어떻게 해야 할까?

```
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

    def sub(self, num):
        self.result -= num
        return self.result
```

클래스와 객체



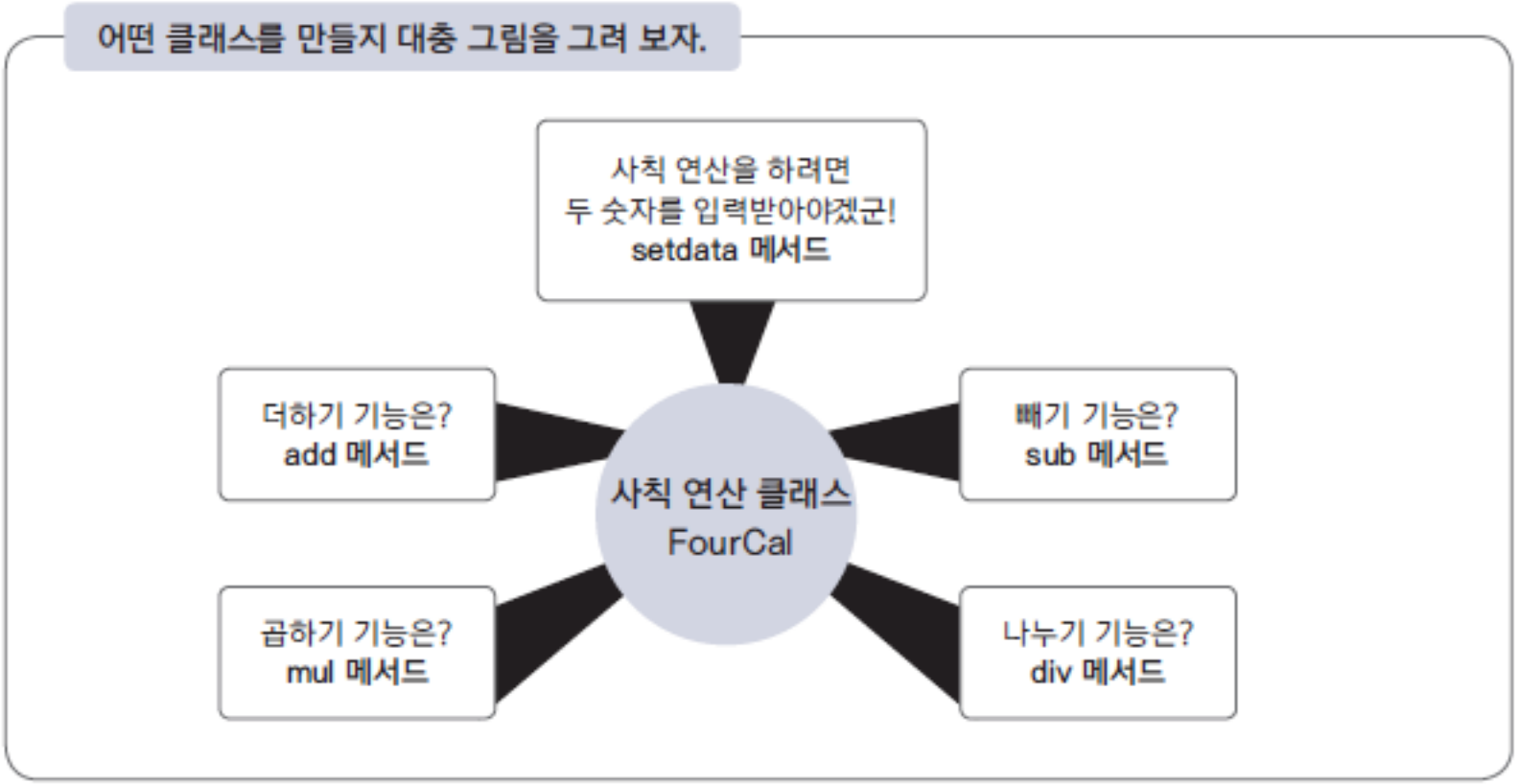
- 클래스(class)란 똑같은 무언가를 계속 만들어 낼 수 있는 설계 도면(과자 틀), 객체(object)란 클래스로 만든 피조물(과자 틀로 찍어 낸 과자)을 뜻함
- 동일한 클래스로 만든 객체들은 서로 영향을 주지 않음

클래스와 객체

- 객체와 인스턴스의 차이
  - a = Cookie()
  - 인스턴스라는 말은 특정 객체(a)가 어떤 클래스(Cookie)의 객체인지를 관계 위주로 설명할 때 사용함
  - 예) 'a는 객체'이며, 'a는 Cookie의 인스턴스'라 불러야함

사칙 연산 클래스 만들기

- 클래스를 어떻게 만들지 먼저 구상하기 (1)



사칙 연산 클래스 만들기

- 클래스를 어떻게 만들지 먼저 구상하기 (1)
  - 사칙 연산 가능을 가진 FourCal 클래스가 다음처럼 동작한다고 가정

  - 먼저 a = FourCal()를 입력해서 a라는 객체를 만듦

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

  - a.setdata(4, 2)를 입력해서 숫자 4와 2를 a에 지정

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

  - a.add()를 수행하면 두 수를 합한 결과(4 + 2)를 출력

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

사칙 연산 클래스 만들기

- 클래스를 어떻게 만들지 먼저 구상하기 (1)
  - 사칙 연산 가능을 가진 FourCal 클래스가 다음처럼 동작한다고 가정

  - a.mul()을 수행하면 두 수를 곱한 결과(4 * 2)를 출력

```
>>> a.mul()
8
```

  - a.sub()를 수행하면 두 수를 뺀 결과(4 - 2)를 출력

```
>>> a.sub()
2
```

  - a.div()를 수행하면 두 수를 나눈 결과(4 / 2)를 출력

```
>>> a.div()
2
```

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)
  - 빈 클래스를 만들고 이름을 'FourCal'로 지정

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

  - pass는 아무것도 수행하지 않는 문법으로, 임시로 코드를 작성할 때 주로 사용

```
>>> a = FourCal()
>>> type(a)
<class '__main__.FourCal'>
```

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)
  - FourCal 클래스에 피연산자를 저장할 수 있는 setdata 메서드 정의

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

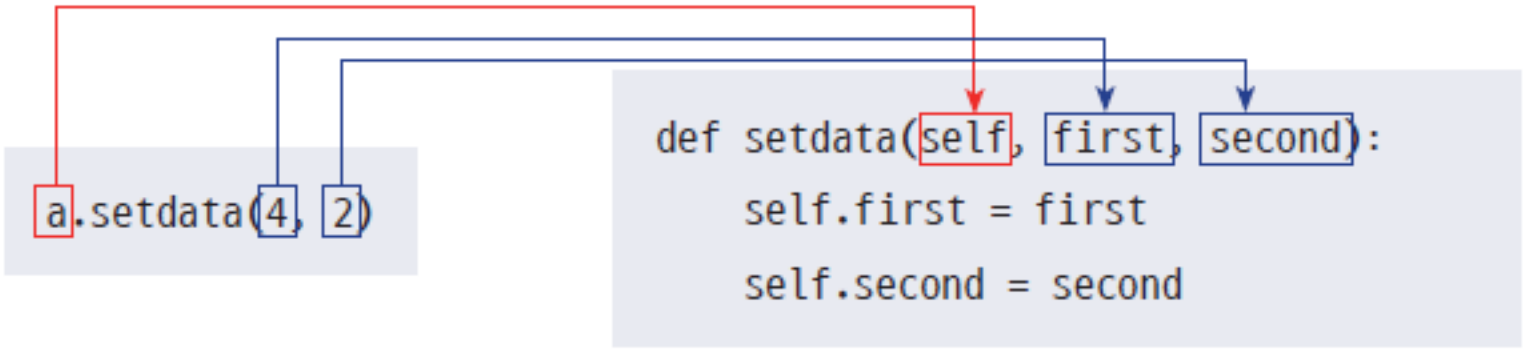
  - 클래스 내 함수를 메서드(method)라 부름
  - 메서드를 정의할 때는 매개변수 앞에 'self'를 적어야함

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)

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

  - setdata 메서드에는 self, first, second 총 3개의 매개변수가 필요한데 실제로는 a.setdata(4, 2)처럼 2개의 값만 전달



사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)

```
def setdata(self, first, second):    # 메서드의 매개변수
    self.first = first               # 메서드의 수행문
    self.second = second             # 메서드의 수행문
```

- a.setdata(4, 2)처럼 호출하면 setdata 메서드의 매개변수 first, second에는 각각 값 4와 2가 전달되어 setdata 메서드의 수행문이 다음과 같이 해석됨

```
self.first = 4
self.second = 2
```

- self는 전달된 객체 a이므로 다시 다음과 같이 해석

```
a.first = 4
a.second = 2
```

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)

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

  - 객체에 생성되는 객체만의 변수(first, second)를 '객체변수' 또는 '속성(attribute)'이라고 부름

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)

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

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

```
>>> b.setdata(3, 7)
>>> b.first
3
```

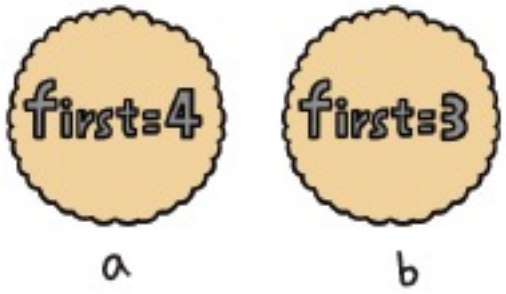
- b객체 선언 후, a 객체의 first에 저장된 값도 3으로 변할까, 아니면 원래대로 값 4를 유지할까?

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)

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

- a 객체의 first 값은 b 객체의 first 값에 영향 받지 않고 원래 값을 유지하고 있다는 것을 확인
- 클래스로 만든 객체의 객체변수는 다른 객체의 객체변수에 상관없이 독립적인 값을 유지



사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)
  - FourCal 클래스에 더하기 기능을 하는 add 메서드 정의

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

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

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

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)
  - 곱하기, 빼기, 나누기 등을 할 수 있도록 프로그램을 개선하시오.

사칙 연산 클래스 만들기

- 클래스 구조 만들기 (2)
  - 곱하기, 빼기, 나누기 등을 할 수 있도록 프로그램을 개선하시오.

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

생성자

- 생성자(constructor)란 객체가 생성될 때 자동으로 호출되는 메서드를 의미
- 파이썬 메서드명으로 '__init__'를 사용하면 이 메서드는 생성자가 됨

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

생성자

- a = FourCal()을 수행할 때 생성자 __init__가 호출되어 오류가 발생

```
>>> a = FourCal()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() missing 2 required positional argument
s: 'first' and 'second'
```

- 오류를 해결하려면 first와 second에 해당하는 값을 전달하여 객체를 생성

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

클래스의 상속

- 상속(Inheritance)이란 '물려받다'라는 뜻으로, '재산을 상속받다'라고 할 때의 상속과 같은 의미
- 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것

- 클래스를 상속하기 위해서는 클래스 이름 뒤 괄호 안에 상속할 클래스 이름을 넣어주면 된다.

```
class 클래스_이름(상속할_클래스_이름)
```

클래스의 상속

- FourCal 클래스를 상속하는 MoreFourCal 클래스는 다음과 같음

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

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

클래스의 상속

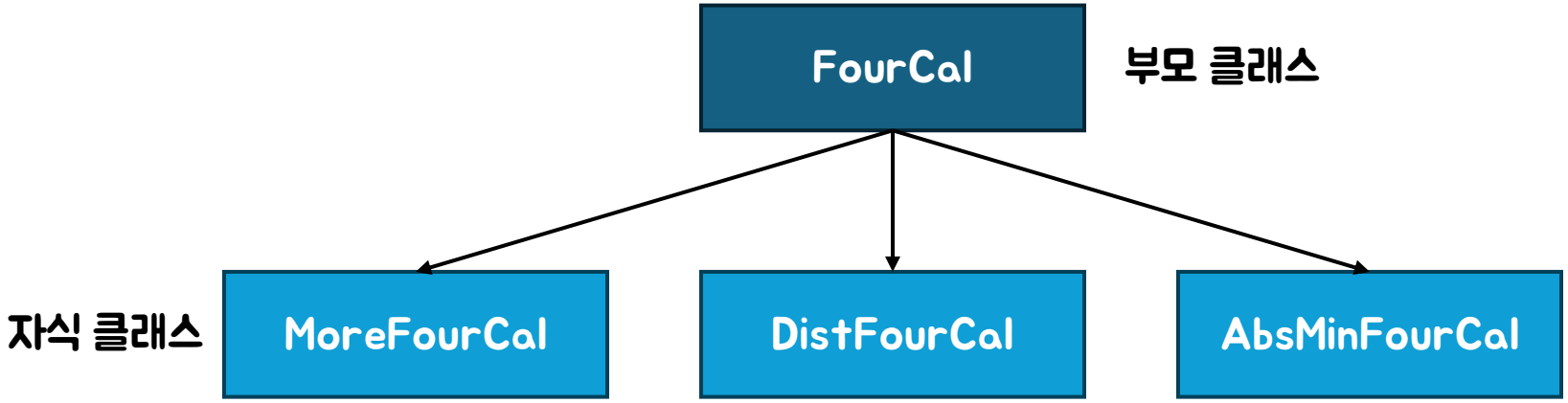
- a^b를 계산하는 MoreFourCal 클래스 정의

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

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

클래스의 상속

- FourCal를 기반으로 다양한 연산들을 만들 수 있음
  - |first-second|^2를 계산하는 DistFourCal 클래스
  - |first|-|second|를 계산하는 AbsMinFourCal 클래스
  - 등등...



- MoreFourCal는 FourCal이다.
- DistFourCal는 FourCal이다.
- AbsMinFourCal는 FourCal이다.

메서드 오버라이딩

```
>>> a = FourCal(4, 0)
>>> a.div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    result = self.first / self.second
ZeroDivisionError: division by zero
```

메서드 오버라이딩

- FourCal 클래스를 상속하는 SafeFourCal 클래스
  - FourCal 클래스에 있는 div 메서드를 동일한 이름으로 다시 작성

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:  # 나누는 값이 0인 경우 0을 리턴하도
록 수정
...             return 0
...         else:
...             return self.first / self.second
```

  - 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것을 메서드 오버라이딩(method overriding)이라고 부름

메서드 오버라이딩

- FourCal 클래스를 상속하는 SafeFourCal 클래스

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:  # 나누는 값이 0인 경우 0을 리턴하도
록 수정
...             return 0
...         else:
...             return self.first / self.second
```

  - FourCal 클래스에 있는 div 메서드를 동일한 이름으로 다시 작성
  - 이렇게 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것을 메서드 오버라이딩(method overriding)이라고 한다.

클래스의 상속

- super 명령어는 자식 클래스에서 부모 클래스의 메소드에 접근할 때 사용함

```
class A:
    def __init__(self):
        print("init in A class")

    def printA(self):
        print("printA in A class")

class B(A):
    def __init__(self):
        super().__init__()
        print("init in B class")

    def printA(self):
        super().printA()
        print("printA in B class")


b = B()
b.printA()
```

클래스 변수

- 클래스 변수는 클래스 안에 메소드를 선언하는 것과 마찬가지로 클래스 안에 변수를 선언하여 생성
- 클래스변수는 객체변수와 달리 클래스로 만든 모든 객체에 공유된다는 특징이 있다.

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

클래스 변수

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

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

- 클래스변수의 값을 변경하면, 클래스로 만든 모든 객체의 클래스 변수들도 변경됨

Q&A