인공지능 기초 프로그래밍

Python 기초 (클래스)

파이썬 프로그램 기초-클래스

- 함수
  - 반복적으로 수행하는 것 중 가치 있는 부분을 하나의 함수로 작성

- 클래스
  - 함수 보다 범용적인 개념으로
    - 큰 틀을 제공하고,
    - 클래스를 그대로 이용하거나 수정 활용 가능

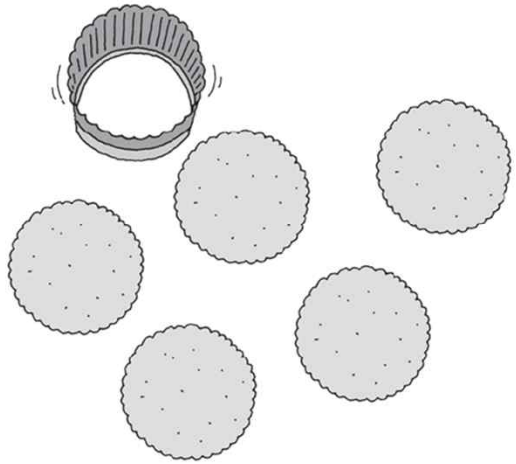
- 함수는 더하기, 빼기, 곱하기 등을 구현
- 클래스는 계산기의 큰 틀을 제공
  - 클래스 내에 메소드(함수)로 더하기, 빼기, 곱하기 등을 만들 수 있음

파이썬 프로그램 기초-클래스

▪ 클래스와 객체

- **클래스(Class)**
  - 똑같은 무엇인가를 계속해서 만들어 낼 수 있는 설계 도면
- **객체(Object)**
  - 클래스로 만든 피조물

> • 과자 틀 → 클래스(class)
> • 과자 틀을 사용해 만든 과자 → 객체(object)



과자 틀(클래스)과 이 틀로 찍어 만든 과자(객체)

파이썬 프로그램 기초-클래스

▪ 클래스와 객체

- 클래스로 만든 객체의 특징
  - 객체마다 고유한 성격을 가짐
  - 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않음

- 파이썬 클래스의 가장 간단한 예

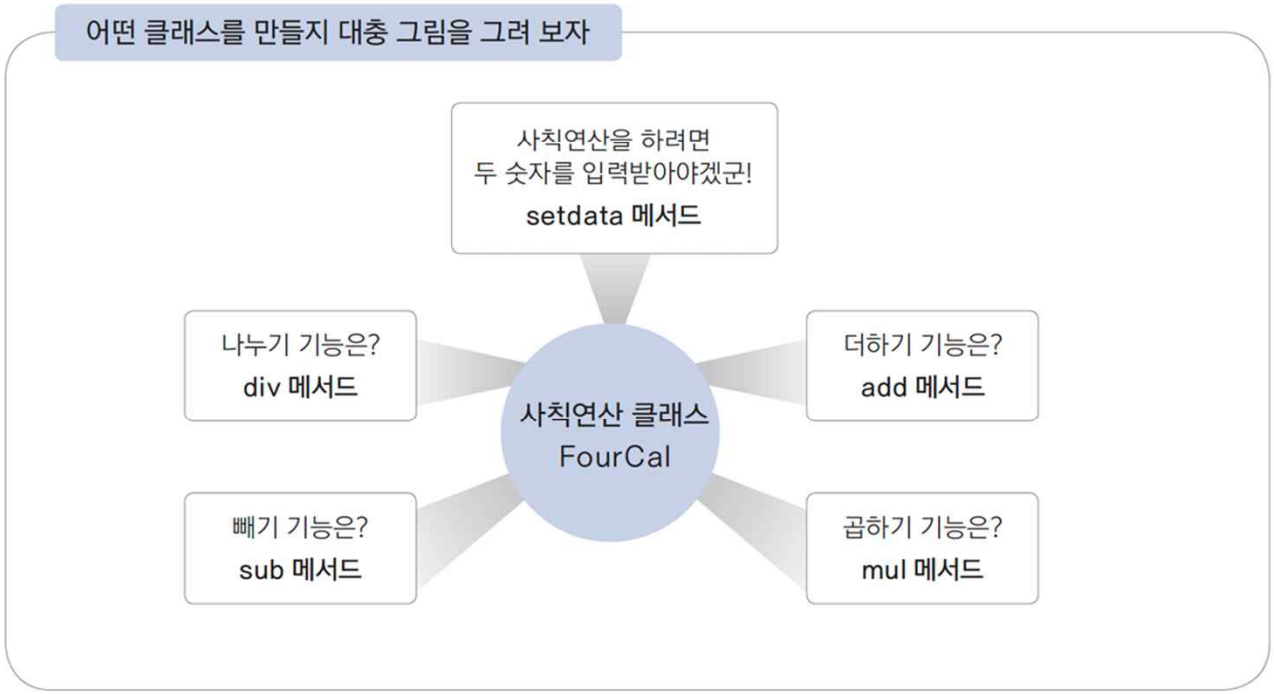
```
>>> class Cookie:
...     pass
...
>>>
```

- Cookie 클래스의 객체를 만드는 방법

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

파이썬 프로그램 기초-클래스

▪ 사칙연산 클래스 구상하기



▪ FourCal 클래스 만들기

- 사칙연산을 가능하게 하는 FourCal 클래스 만들기

1. 클래스 구조 만들기

- pass란 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.
```


▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

- 더하기 · 나누기 · 곱하기 · 빼기 등의 기능을 하는 객체 만들기
- 우선 객체에 사칙연산을 할 때 사용할 2개의 숫자를 알려주어야 함
- pass 문장을 삭제하고 setdata 함수 생성

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

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


▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

- **메서드(Method)**
  - 클래스 안에 구현된 함수

- 일반적인 함수

```
def 함수 이름(매개변수):
    수행할 문장
    ...
```

- setdata 메서드

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

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

▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

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

① 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
  - setdata 메서드에는 총 3개의 매개변수가 필요한데 실제로는 2개의 값만 전달하는 이유는?

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

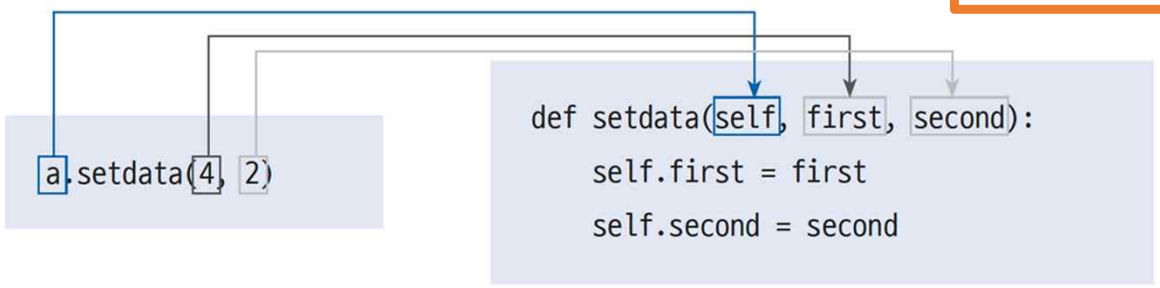

▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

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

① setdata 메서드의 매개변수

- setdata 메서드의 첫 번째 매개변수 **self**에는 setdata 메서드를 호출한 **객체 a가 자동으로 전달**되기 때문



▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

```
self.first = first
self.second = second
```
② 메서드의 수행문

② setdata 메서드의 수행문

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

```
self.first = 4
self.second = 2
```
=
self는 전달된 객체 a
```
a.first = 4
a.second = 2
```

▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

```
self.first = first
self.second = second
```
← ② 메서드의 수행문

② setdata 메서드의 수행문

- a 객체에 객체변수 first와 second가 생성되고 지정된 값이 저장됨

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


▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

- 객체의 객체변수 특징 살펴보기
  - a, b 객체 생성

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

  - a 객체의 객체변수 first 생성

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

  - b 객체의 객체변수 first 생성

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

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

▪ FourCal 클래스 만들기

2. 객체에 숫자 지정할 수 있게 만들기

- 객체의 객체변수 특징 살펴보기
  - a 객체의 first 값은 b 객체의 first 값에 영향 받지 않고 원래 값을 유지!

```
>>> print(a.first)
4
```

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

- id 함수를 사용하여 증명
  - 객체변수는 그 객체의 고유 값을 저장하는 공간

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 7)
>>> id(a.first) ◀── a의 first 주소 값을 확인
1839194944
>>> id(b.first) ◀── b의 first 주소 값을 확인
1839194928
```


▪ FourCal 클래스 만들기

3. 더하기 기능 만들기

- 클래스에 2개의 숫자를 더하는 add 메서드 추가

```
>>> 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)
```

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


▪ FourCal 클래스 만들기

3. 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 = a.first + a.second
```

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

```
result = 4 + 2
```

- 결과

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


▪ FourCal 클래스 만들기

4. 곱하기 · 빼기 · 나누기 기능 만들기

- add 메서드와 동일한 방법으로 mul, sub, div 메서드 생성

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

```
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2
```


▪ 생성자(Constructor)

- 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'
```


▪ 생성자(Constructor)

▪ 생성자(Constructor)

- 객체가 생성될 때 자동으로 호출되는 메서드
- 메서드 이름으로 __init__ 사용 예약어[수정 불가능]

```
>>> 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)
```

매개변수	값
self	생성되는 객체
first	4
second	2

▪ 클래스 상속

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

```
class 클래스 이름(상속할 클래스 이름)
```

- FourCal 클래스를 상속하는 MoreFourCal 클래스

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

▪ 클래스 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
```

파이썬 프로그램 기초-클래스

- 상속 (Inheritance)
  - 기존 클래스를 변경하지 않고 **기능을 추가하거나 기존 기능을 변경**하고 싶을 때 사용
    - 기존 클래스를 수정?
      - 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이 있음
      - 클래스 사용 이유...(큰 틀 제공!!)

```
class MoreCalc(FourCalc):
    # Inheritance
    # Can add new Method
    def pow(self):
        return self.first ** self.second

    # Overriding: Can modified method from inheritance method
    def div(self):
        if self.second == 0:
            return 0
        else:
            return self.first / self.second
```

상속을 통하여 기존 사칙연산 +
거듭제곱이 가능한 메소드를 추가

상속을 통하여 메서드 오버라이딩

▪ 클래스 변수

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

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

```
>>> print(Family.lastname)
김
```


▪ 클래스 변수

- Family 클래스로 만든 객체를 통해서도 클래스 변수 사용 가능

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

- Family 클래스의 lastname을 변경하면?

```
>>> Family.lastname = "박"
```

- 클래스로 만든 객체의 lastname 값도 모두 변경됨

```
>>> print(a.lastname)
박
>>> print(b.lastname)
박
```


▪ 클래스 변수

- id 함수를 통한 클래스 변수 확인
  - 클래스로 만든 모든 객체의 클래스 변수가 모두 같은 메모리를 가리키고 있음

```
>>> id(Family.lastname)
4480159136
>>> id(a.lastname)
4480159136
>>> id(b.lastname)
4480159136
```

- **클래스 변수는 클래스로 만든 모든 객체에 공유된다는 특징이 있음**

Practice #1 ()

```python
class FourCalc:
    def set_data(self, first, second):
        self.first = first
        self.second = second

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

    def add(self):
        return self.first + self.second

    def mul(self):
        return self.first * self.second

    def div(self):
        return self.first / self.second


calc_0 = FourCalc(4, 2)
calc_1 = FourCalc(2, 2)
calc_2 = FourCalc(4, 0)

print(calc_0.add(), calc_1.add(), calc_0.mul(), calc_1.mul())
# print(calc_2.div())
```

```python
class MoreCalc(FourCalc):
    # Inheritance
    # Can add new Method
    def pow(self):
        return self.first ** self.second

    # Overriding: Can modified method from inheritance method
    def div(self):
        if self.second == 0:
            return 0
        else:
            return self.first / self.second


moreCalc_0 = MoreCalc(4, 2)
moreCalc_1 = MoreCalc(2, 2)
moreCalc_2 = MoreCalc(4, 0)

print(moreCalc_0.add(), moreCalc_1.add(), moreCalc_0.mul(), moreCalc_1.mul(),
      moreCalc_0.pow(), moreCalc_1.pow(), moreCalc_2.div())
```

Thank you

Q&A

www.kopo.ac.kr
jsshin7@kopo.ac.kr

