

세상의 속도를
따라잡고 싶다면



점프 투 파이썬

박응용 지음(위키독스 운영자)



파이썬 날개 달기



05-1 클래스



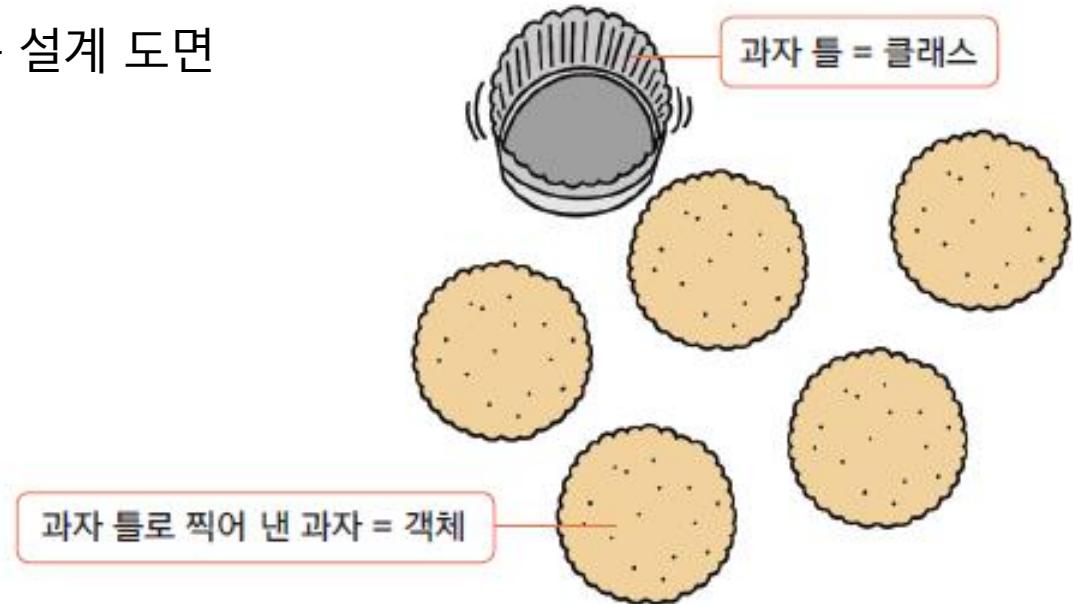
■ 클래스와 객체

■ 클래스(class)

- 똑같은 무언가를 계속 만들어 낼 수 있는 설계 도면

■ 객체(object)

- 클래스로 만든 피조물



■ 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
```

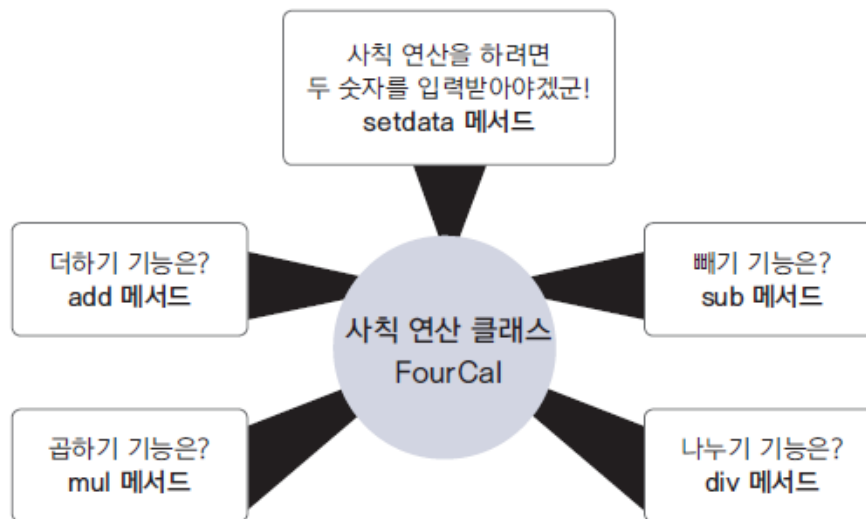
- Cookie 클래스의 객체를 만드는 방법

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

■ 사칙연산 클래스 만들기

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

어떤 클래스를 만들지 대충 그림을 그려 보자.



■ 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.
```

■ 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
```

■ 사칙연산 클래스 만들기

■ 객체에 연산할 숫자 지정하기

■ 메서드(method)

- 클래스 안에 구현된 함수

■ 일반적인 함수

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

■ 한 줄에 결괏값 출력하기

```
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

■ 사칙연산 클래스 만들기

■ 객체에 연산할 숫자 지정하기

```
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

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

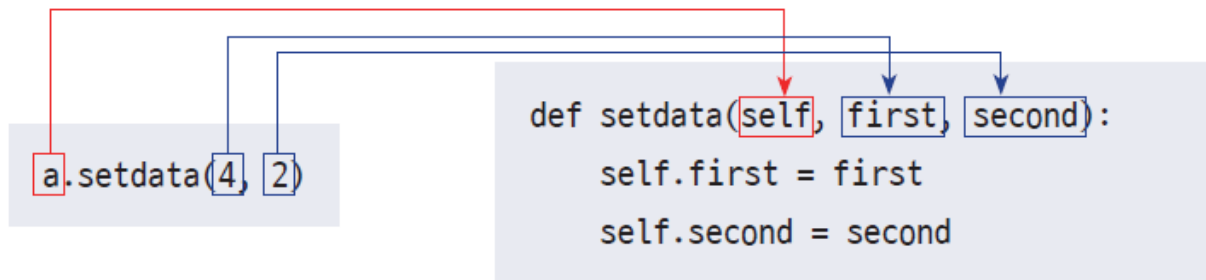
■ 사칙연산 클래스 만들기

■ 객체에 연산할 숫자 지정하기

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

① 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

■ 사칙연산 클래스 만들기

■ 객체에 연산할 숫자 지정하기

```
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)  
>>> a.first  ← a 객체의 first 변수값 출력  
4  
>>> a.second ← a 객체의 second 변수값 출력  
2
```

■ 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
```

■ 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!



■ 사칙연산 클래스 만들기

- 더하기 기능 만들기
 - 클래스에 2개의 숫자를 더하는 add 메서드 추가

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

두 숫자를 더하는 add
메서드를 만들면 되겠다!



```
>>> 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
```

■ 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

```
result = 4 + 2
```

■ 결과

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

■ 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
```


■ 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'
```

■ 생성자

■ 생성자(constructor)

- 객체가 생성될 때 자동으로 호출되는 메서드
- 메서드 명으로 `__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
```

■ 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)
```

- FourCal 클래스를 상속하는 MoreFourCal 클래스

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

- 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
```

■ 클래스의 상속

- a^b 를 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
```

■ 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
```

■ 클래스변수

- 다른 객체들에 영향 받지 않고 독립적으로 값을 유지하는 객체변수와는 다름
- 클래스 안에 변수를 선언하여 생성
 - 예) Family 클래스에 클래스 변수 lastname 선언

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

```
>>> Family.lastname  
김
```

■ 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



*“Life is too short,
You need Python!”*

인생은 너무 짧으니,
파이썬이 필요해!

감사합니다.