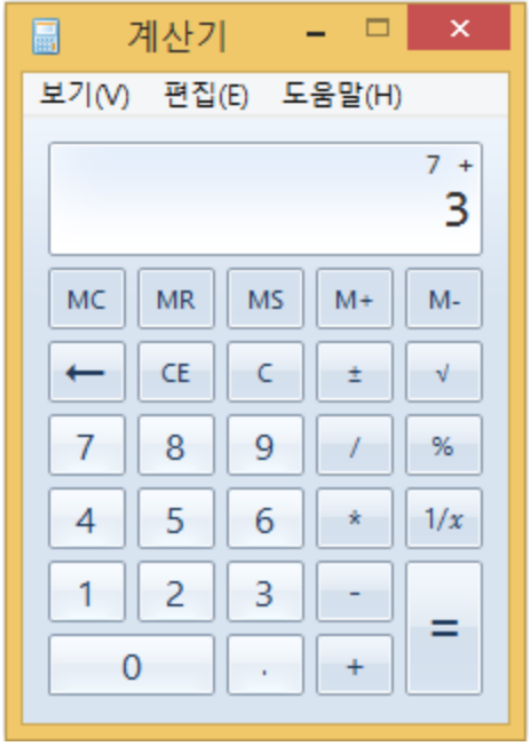
프로그래밍 기초

파이썬 기초 클래스_모듈

클래스

클래스

클래스란




```
result = 0

def add(num):
    global result
    result += num
    return result

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

```
3
7
```

클래스

클래스란

```
result1 = 0
result2 = 0

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

def add2(num):
    global result2
    result2 += num
    return result2

print(add1(3))
print(add1(4))
print(add2(3))
print(add2(7))
```

```
3
7
3
10
```

클래스

클래스란

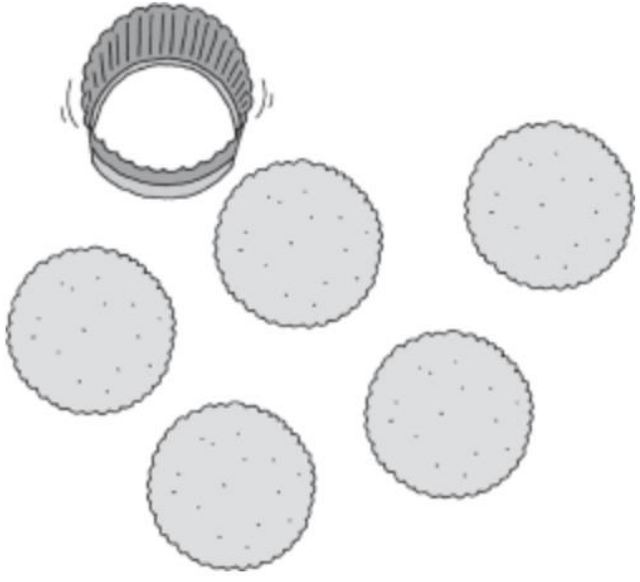
```
class Calculator:
    def __init__(self):
        self.result = 0

    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))
```

```
3
7
3
10
```

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

클래스

클래스와 객체



클래스

클래스란

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

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

클래스

클래스란

[객체와 인스턴스의 차이]

클래스로 만든 객체를 인스턴스라고도 한다. 그렇다면 객체와 인스턴스의 차이는 무엇일까? 이렇게 생각해 보자. a = Cookie() 이렇게 만든 a는 객체이다. 그리고 a 객체는 Cookie의 인스턴스이다. 즉 인스턴스라는 말은 특정 객체(a)가 어떤 클래스(Cookie)의 객체인지를 관계 위주로 설명할 때 사용한다. "a는 인스턴스"보다는 "a는 객체"라는 표현이 어울리며 "a는 Cookie의 객체"보다는 "a는 Cookie의 인스턴스"라는 표현이 훨씬 잘 어울린다.

클래스

사칙연산 클래스 만들기

클래스를 어떻게 만들지 먼저 구상하기

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

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

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

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

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

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

클래스

클래스 구조 만들기

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

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

클래스

객체에 숫자 지정할 수 있게 만들기

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

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

클래스

객체에 숫자 지정할 수 있게 만들기

```
def 함수명(매개변수):
    수행할 문장
    ...
```

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

클래스

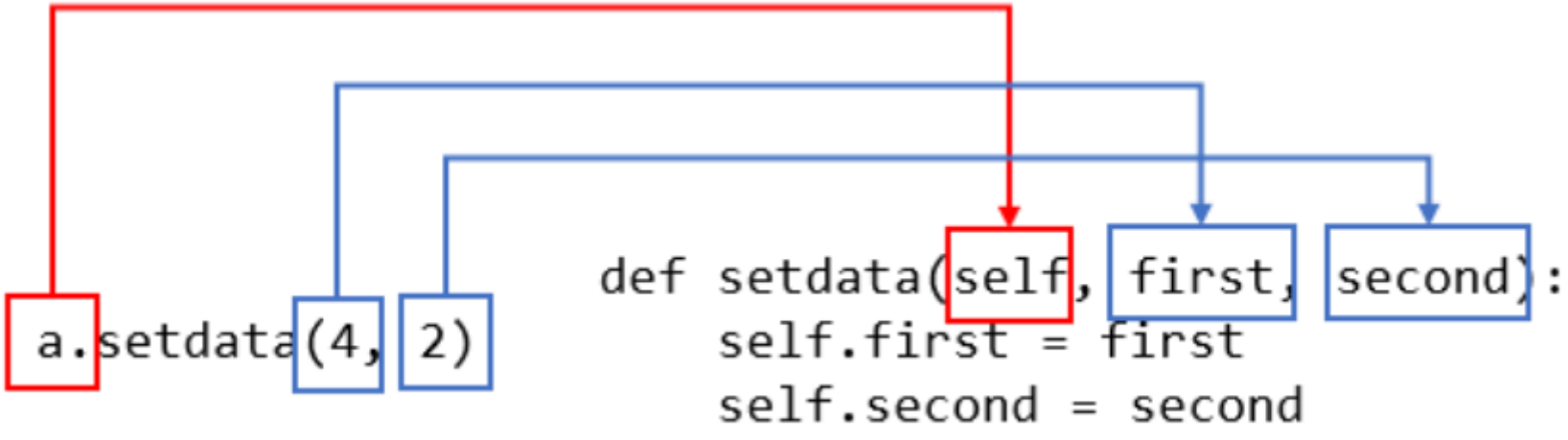
객체에 숫자 지정할 수 있게 만들기

① setdata 메서드의 매개변수

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

클래스

객체에 숫자 지정할 수 있게 만들기



클래스

객체에 숫자 지정할 수 있게 만들기

[메서드의 또 다른 호출 방법]

잘 사용하지는 않지만 다음과 같이 클래스를 통해 메서드를 호출하는 것도 가능하다.

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

위와 같이 `클래스 이름.메서드` 형태로 호출할 때는 객체 a를 첫 번째 매개변수 self에 꼭 전달해 주어야 한다. 반면에 다음처럼 `객체.메서드` 형태로 호출할 때는 self를 반드시 생략해서 호출해야 한다.

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

클래스

객체에 숫자 지정할 수 있게 만들기

② setdata 메서드의 수행문

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

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

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

클래스

객체에 숫자 지정할 수 있게 만들기

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

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

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

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

클래스

객체에 숫자 지정할 수 있게 만들기

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

클래스

객체에 숫자 지정할 수 있게 만들기

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 7)
>>> id(a.first)   # a의 first 주소값을 확인
1839194944
>>> id(b.first)   # b의 first 주소값을 확인
1839194928
```

클래스

객체에 숫자 지정할 수 있게 만들기

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

클래스

더하기 기능 만들기

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

클래스

더하기 기능 만들기

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

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

클래스

더하기 기능 만들기

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

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

```
result = a.first + a.second
```

```
result = 4 + 2
```

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

클래스

곱하기, 빼기, 나누기 기능 만들기

```
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
```

클래스

곱하기, 빼기, 나누기 기능 만들기

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 8)
```

```
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2
>>> b.add()
11
>>> b.mul()
24
>>> b.sub()
-5
>>> b.div()
0.375
```

클래스

생성자

```
>>> 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'
```

클래스

생성자

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

클래스

생성자

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

클래스

생성자

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

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

클래스

생성자

매개변수	값
self	생성되는 객체
first	4
second	2

클래스

생성자

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

```
>>> a = FourCal(4, 2)
>>> a.add()
6
>>> a.div()
2.0
```

클래스

클래스의 상속

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

class 클래스 이름(상속할 클래스 이름)

클래스

클래스의 상속

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

클래스

클래스의 상속

왜 상속을 해야 할까?

보통 상속은 기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경하려고 할 때 사용한다.

"클래스에 기능을 추가하고 싶으면 기존 클래스를 수정하면 되는데 왜 굳이 상속을 받아서 처리해야 하지?" 라는 의문이 들 수도 있다. 하지만 기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이라면 상속을 사용해야 한다.

클래스

클래스의 상속

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

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

클래스

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
```

클래스

메서드 오버라이딩

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

클래스

메서드 오버라이딩

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

클래스

클래스 변수

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

```
>>> print(Family.lastname)
김
```

클래스

메서드 오버라이딩

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

```
>>> Family.lastname = "박"
```

클래스

메서드 오버라이딩

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

클래스

메서드 오버라이딩

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

모듈

모듈

모듈 만들기

```
# mod1.py
def add(a, b):
    return a + b

def sub(a, b):
    return a-b
```

모듈

모듈 불러오기

```
>>> import mod1
>>> print(mod1.add(3, 4))
7
>>> print(mod1.sub(4, 2))
2
```

모듈

모듈 만들기

```
import 모듈이름
```

> from 모듈이름 import 모듈함수

```
>>> from mod1 import add
>>> add(3, 4)
7
```

```
from mod1 import add, sub
```

```
from mod1 import *
```

모듈

if __name__ == "__main__": 의 의미

```
# mod1.py
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

print(add(1, 4))
print(sub(4, 2))
```

```
print(add(1, 4))
print(sub(4, 2))
```

모듈

if __name__ == "__main__": 의 의미

```python
# mod1.py
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

모듈

if __name__ == "__main__": 의 의미

`__name__` 변수란?

파이썬의 `__name__` 변수는 파이썬이 내부적으로 사용하는 특별한 변수 이름이다. 만약 `C:\doit>python mod1.py` 처럼 직접 mod1.py 파일을 실행할 경우 mod1.py의 `__name__` 변수에는 `__main__` 값이 저장된다. 하지만 파이썬 셸이나 다른 파이썬 모듈에서 mod1을 import 할 경우에는 mod1.py의 `__name__` 변수에는 mod1.py의 모듈 이름 값 mod1이 저장된다.

모듈

클래스나 변수 등을 포함한 모듈

```
# mod2.py
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

def add(a, b):
    return a+b
```

```
>>> a = mod2.Math()
>>> print(a.solv(2))
12.566368
```

```
>>> print(mod2.add(mod2.PI, 4.4))
7.541592
```

모듈

다른 파일에서 모듈 블러오기

```
# modtest.py
import mod2
result = mod2.add(3, 4)
print(result)
```