

06. 파이썬 클래스

이번 장에서는 파이썬 클래스(class)를 배워보겠습니다. 사실 기본적인 프로그램을 작성하는 데는 지금까지 배운 변수, 기본 자료구조, 제어문, 함수 정도면 충분합니다. 즉, 이번 장에서 배울 클래스를 모르더라도 기본적인 프로그램 정도는 작성할 수 있습니다. 참고로 C 언어처럼 개발된 지 오래된 프로그래밍 언어는 언어 자체에서 클래스를 지원하지 않는 경우도 많습니다.

클래스를 지원하지 않는 C 언어와 달리 1990년대 이후에 개발된 C++, 자바(Java), C# 같은 프로그래밍 언어는 기본적으로 클래스를 지원합니다. 최근에는 클래스를 이용한 객체 지향 프로그래밍이 컴퓨터 프로그래밍 분야에서 매우 기본적인 개발 패러다임으로 자리 잡았습니다. 객체 지향 프로그래밍을 사용하면 우리가 일상생활에서 생각하는 것과 비슷한 논리로 프로그램을 작성할 수 있기 때문에 프로그램 개발과 유지보수가 간편해집니다. 이러한 장점 때문에 객체 지향 프로그래밍은 대규모의 프로그램을 개발할 때 자주 사용됩니다.

1) 클래스란?

지금까지 클래스라는 개념을 몰랐어도 프로그래밍을 하는 데 큰 어려움이 없었습니다. 그러나 이 책의 뒷부분에서 살펴볼 증권사(대신증권, 이베스트증권, 키움증권) API 를 사용하려면 반드시 클래스라는 개념을 잘 알아야 합니다. 왜냐하면 국내 증권사에서 제공하는 API 가 대부분 클래스를 이용한 객체 지향 프로그래밍 방식으로 개발됐기 때문입니다.

물론, 이처럼 직접적인 이유가 아니더라도 클래스를 이용해 프로그래밍하면 데이터와 데이터를 조작하는 함수를 하나의 묶음으로 관리할 수 있으므로 복잡한 프로그램도 더욱 쉽게 작성할 수 있습니다. 간단한 예제를 통해 파이썬의 클래스에 대해 본격적으로 배워보겠습니다.

여러분이 명함을 제작하는 스타트업을 창업했다고 생각해봅시다. 이 스타트업은 고객의 명함을 제작하기에 앞서 명함에 들어갈 고객 정보(이름, 이메일 주소, 근무지 주소)를 프로그램을 통해 입력받습니다. 파이썬 변수를 이용하여 다음과 같이 값을 저장할 수 있습니다.

```
>>> name = "kimyuna"
```

```
>>> email = "yunakim@naver.com"
```

```
>>> addr = "seoul"
```

```
>>>
```

다음으로 입력받은 데이터를 사용해 실제로 명함을 출력하는 함수를 하나 만들어 보겠습니다. 다음과 같이 이름, 이메일 주소, 근무지 주소를 함수 인자로 입력받아 포맷에 맞춰 입력 값을 출력합니다.

```
>>> def print_business_card(name, email, addr):
```

```
    print("-----")
```

```
    print("Name: %s" % name)
```

```
    print("E-mail: %s" % email)
```

```
    print("Office Address: %s" % addr)
```

```
    print("-----")
```

```
>>>
```

작성한 함수가 잘 동작하는지 테스트하기 위해 다음과 같이 함수에 인자값을 넣어 함수를 호출해보기 바랍니다. 참고로 함수의 인자로 사용된 변수들은 입력된 값을 바인딩하고 있습니다. 결과값을 보니 여러분이 작성한 명함 출력 함수가 제대로 동작하는 것을 알 수 있습니다.

```
>>> print_business_card(name, email, addr)
```

```
-----
```

```
Name: kimyuna
```

```
E-mail: yunakim@naver.com
```

```
Office Address: seoul
```

```
-----
```

```
>>>
```

여러분이 창업한 스타트업은 어느덧 입소문을 타서 드디어 회원 수가 한 명에서 두 명으로 늘어났습니다. 회원이 한 명뿐이었던 시절에는 고객 정보를 바인딩할 변수로 name, email, addr 이라는 이름을 이용했습니다. 이제 회원 수가 한 명 더 늘어서 두 번째 회원에 대한 입력 정보는 다음과 같이 새로운 변수를 사용해 저장했습니다. 참고로 같은 변수를 사용하면 첫 번째 회원에 대한 정보가 손실된다는 것은 아시겠죠?

```
>>> name2 = "DusanBack"
```

```
>>> email2 = "dusan.back@naver.com"
```

```
>>> addr2 = "Kyunggi"
```

```
>>>
```

두 번째 회원의 고객 정보도 잘 입력받았으므로 명함을 출력해보겠습니다. 앞서 명함을 출력하기 위한 print_business_card 라는 이름의 함수를 작성해 뒀기 때문에 두 번째 회원의 명함을 출력하려면 기존에 작성한 명함 출력 함수에 적절한 입력값만 전달하고 함수를 호출하면 됩니다.

```
>>> print_business_card(name2, email2, addr2)
```

```
-----
```

```
Name: DusanBack
```

```
E-mail: dusan.back@naver.com
```

Office Address: Kyunggi

>>>

아직은 회원이 두 명이라 걱정이 없는데 앞으로 회원 수가 늘어나면 개인 정보를 어떤 방식으로 저장하는 것이 좋을까요? 그리고 고객 정보에 전화번호 및 팩스 번호를 추가로 저장하고 이를 출력하는 기능을 추가해야 한다면 어떻게 프로그램을 변경하는 것이 좋을지 생각해 봅시다.

1-1) 클래스 기초

지금까지 작성했던 프로그램을 생각해보면 보통 어떤 데이터가 존재하고 데이터를 조작하는 함수를 통해 데이터를 목적에 맞게 변경한 후 이를 다시 출력하는 형태였습니다. 앞서 살펴본 명함 출력 프로그램도 사용자로부터 개인정보를 입력받은 후 해당 데이터를 조작하고 이를 출력하는 함수로 구성돼 있었습니다.

명함 출력에 사용된 이름, 이메일, 주소 등을 효과적으로 관리하기 위해 보통 그림 6.1 과 같이 각 데이터에 대해 별도의 리스트 자료 구조를 사용합니다. 그림 6.1 과 같은 자료 구조에서 첫 번째 회원에 대한 개인 정보를 얻기 위해서는 `name_list[0]`, `email_list[0]`, `address_list[0]`과 같이 각 리스트별로 인덱싱을 수행해야 합니다. 참고로 그림 6.1 과 같이 데이터와 데이터를 처리하는 함수가 분리돼 있고 함수를 순차적으로 호출함으로써 데이터를 조작하는 프로그래밍 방식을 절차 지향 프로그래밍이라고 합니다.

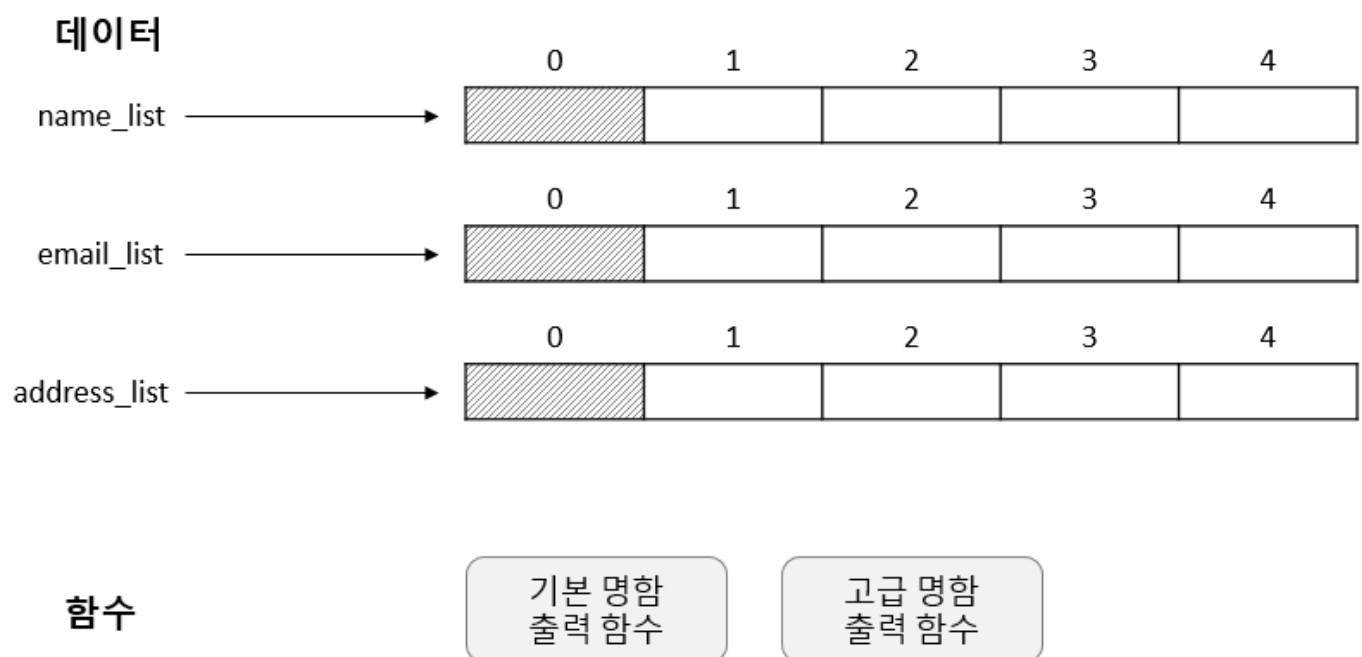


그림 6.1 데이터와 함수

절차 지향 프로그래밍 방식과 달리 객체 지향 프로그래밍은 객체를 정의하는 것으로부터 시작합니다. 앞서 예시로 사용한 명함 스타트업을 통해 객체 지향 프로그래밍에서 말하는 객체라는 개념을 배워봅시다.

먼저 명함을 구성하는 데이터를 생각해보면 명함에는 이름, 이메일, 주소라는 값이 있습니다. 다음으로 명함과 관련된 함수를 생각해보면 기본 명함을 출력하는 함수와 고급 명함을 출력하는 함수가 있을 것입니다. 객체 지향 프로그래밍이란 그림 6.2 와 같이 명함을 구성하는 데이터와 명함과 관련된 함수를 묶어서 명함이라는 새로운 객체(타입)를 만들고 이를 사용해 프로그래밍하는 방식을 의미합니다.

명함

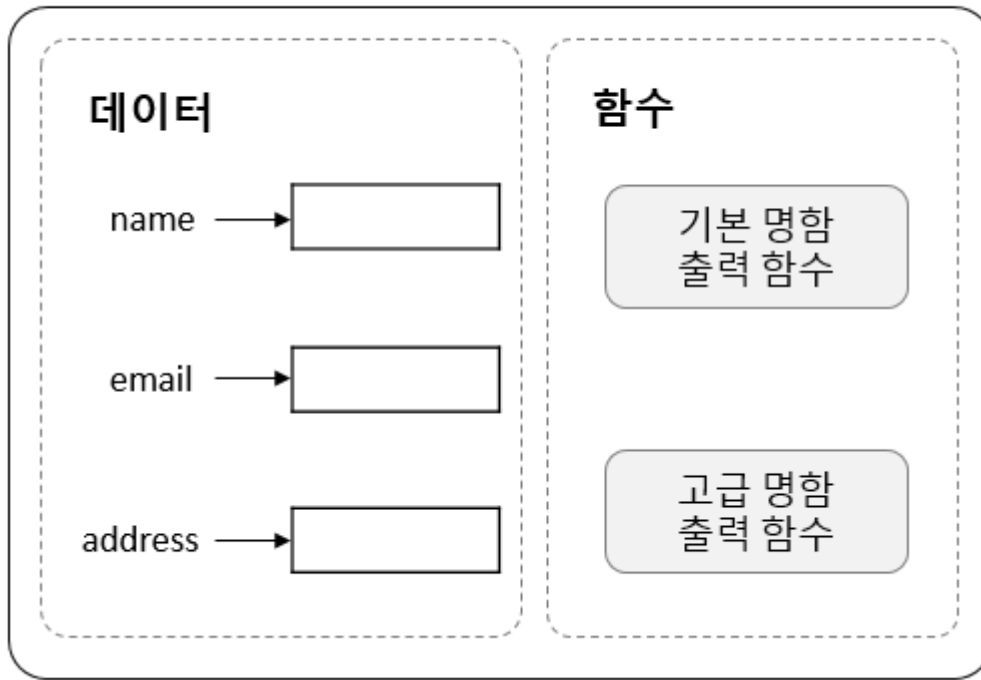


그림 6.2 명함 객체 정의

파이썬을 통해 명함이라는 객체를 만들어 두면 명함 객체가 정수, 실수, 문자열과 마찬가지로 하나의 타입으로 인식되기 때문에 그림 6-3 과 같이 여러 명의 명함 정보도 쉽게 관리할 수 있습니다(이전에는 그림 6.1 처럼 이름, 이메일, 주소가 각각 따로 관리되는 형태였습니다). 마치 다섯 개의 정숫값을 변수로 바인딩하는 것처럼 다섯 개의 명함 값을 변수로 바인딩하거나 리스트에 명함이라는 객체를 저장하고 있으면 되는 것입니다.

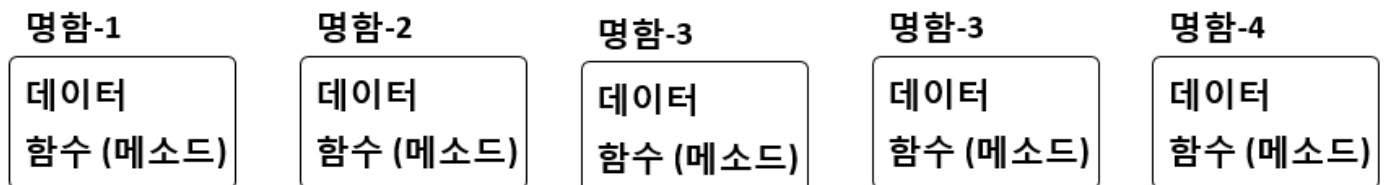


그림 6.3 클래스 객체를 이용해 여러 데이터를 표현

클래스를 이용하면 새로운 타입을 만들 수 있다고 했는데, 사실 앞서 배운 정수, 실수, 문자열, 리스트, 튜플과 같은 기본 자료형과 기본 자료구조도 모두 클래스를 통해 만들어진 타입입니다. 다만 여러분이 만든 것이 아니라 이미 만들어져 있었다는 차이점만 있습니다.

다음 코드를 실행해 여러 파이썬 객체에 대한 반환 타입을 보면 각 타입 앞에 `class` 라는 키워드가 있음을 확인할 수 있습니다.

```
>>> type(3)
```

```
<class 'int'>
```

```
>>> type(3.1)
```

```
<class 'float'>
```

```
>>> type('3')
```

```
<class 'str'>
```

```
>>> type([])
```

```
<class 'list'>
```

```
>>> type()
```

```
<class 'tuple'>
```

```
>>>
```

다른 프로그래밍 언어를 배운 분들에게는 조금 생소하겠지만 파이썬에서는 함수도 객체입니다. 다음과 같이 foo 라는 함수를 정의한 후 해당 함수의 type 을 확인해보니 역시 class 라는 키워드가 붙어 있음을 확인할 수 있습니다.

```
>>> def foo():
```

```
    pass
```

```
>>> type(foo)
```

```
<class 'function'>
```

1-2) 클래스 정의

파이썬에서 함수를 정의한 것처럼 클래스 역시 먼저 정의해야 합니다. 파이썬 IDLE 를 실행한 후 다음과 같이 코드를 입력해 봅시다.

```
>>> class BusinessCard:
```

```
    pass
```

```
>>>>
```

파이썬에서 함수를 정의할 때 def 라는 키워드를 썼던 것처럼 파이썬에서 클래스를 정의하려면 class 라는 키워드를 사용합니다. 클래스를 사용하는 목적이 변수와 함수를 묶어서 하나의 새로운 객체(타입)으로 만드는 것이기 때문에 당연히 클래스에 변수나 함수를 포함시켜 클래스를 정의할 수 있습니다. 다만 위 코드에서는 가장 간단한 형태의 클래스 정의를 보여주기 위해 변수나 함수를 넣지 않고 pass 라는 키워드만을 사용했습니다. pass 키워드를 사용하면 클래스 내부에 아무것도 넣지 않은 상태로 클래스를 정의할 수 있습니다.

파이썬에서 클래스를 정의한다는 것은 새로운 데이터 타입을 정의한 것이기 때문에 이를 실제로 사용하려면 인스턴스라는 것을 생성해야 합니다. 클래스와 인스턴스의 관계를 봉어빵을 만드는 작업으로 비유해 보면 클래스를 정의하는 것은 봉어빵을 만들 때 사용하는 기본 틀을 제작하는 것에 해당합니다. 잉어빵, 황금 봉어빵과 같은 차별화된 봉어빵을 만들기 위해 틀을 잘 정의하는 것이지요. 그러나 아무리 틀을 예쁘고 정교하게 만들었다고 해서 봉어빵이 나오는 것은 아닙니다. 실제로 먹을 수 있는 봉어빵을 만들려면 봉어빵 틀에 반죽을 넣고 봉어빵을 구워야 합니다. 이처럼 봉어빵 틀에 반죽을 넣어서 만들어진 봉어빵이 인스턴스에 해당합니다.

파이썬에서 정의된 클래스를 이용해 인스턴스를 생성하려면 다음과 같이 클래스 이름 뒤에 ()를 넣으면 됩니다. 어떻게 보면 인스턴스를 생성하는 과정은 함수를 호출하는 것과 비슷해 보입니다. 다음 코드에서 첫 번째 문장이 실행되면 그림 6.4 와 같이 'BusinessCard'라는 클래스의 인스턴스가 메모리의 0x0302ABB0 위치에 생성되고 card1 이라는 변수가 이를 바인딩하게 됩니다.

```
>>> card1 = BusinessCard()
```

```
>>> card1
```

```
<__main__.BusinessCard object at 0x0302ABB0>
```

```
>>>
```




그림 6.4 클래스의 인스턴스 생성

생성된 인스턴스의 타입을 살펴보면 조금 전 정의했던 `BusinessCard` 클래스임을 확인할 수 있습니다. 앞서 설명해드린 것처럼 파이썬의 `class` 키워드를 사용하면 원하는 객체(타입)를 만들 수 있음을 확인할 수 있습니다.

```
>>> type(card1)
```

```
<class '__main__.BusinessCard'>
```

```
>>>
```

1-3) 클래스에 메서드 추가하기

앞서 정의한 BusinessCard 클래스는 클래스 내부에 변수나 함수가 없었습니다. 그래서 인스턴스를 만들었음에도 해당 인스턴스로 할 수 있는 일이 별로 없었습니다. 이번에는 BusinessCard 클래스에 사용자로부터 데이터를 입력받고 이를 저장하는 기능을 하는 함수를 추가해보겠습니다. 참고로 클래스 내부에 정의돼 있는 함수를 특별히 메서드(method)라고 부릅니다.

다음 코드는 BusinessCard 클래스에 set_info()라는 메서드를 추가한 것입니다. 메서드 정의는 함수 정의와 마찬가지로 def 키워드를 사용합니다. set_info 메서드는 네 개의 인자를 받는데, 그중 name, email, addr 은 사용자로부터 입력받은 데이터를 메서드로 넘길 때 사용하는 인자입니다. 그렇다면 메서드의 첫 번째 인자인 self 는 무엇일까요?

```
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr
```

```
>>>
```

파이썬 클래스에서 self의 의미를 정확히 이해하는 것이 중요하지만 아직 제대로 설명하기는 조금 이른 감이 있습니다. 일단 클래스에 내부에 정의된 함수인 메서드의 첫 번째 인자는 반드시 self여야 한다고 당분간만 외우고 사용하기 바랍니다.

위 코드에서 메서드 내부를 살펴보면 메서드 인자로 전달된 name, email, addr 값을 self.name, self.email, self.addr이라는 변수에 대입하는 것을 볼 수 있습니다. 앞서 여러 번 설명한 것처럼 파이썬에서 대입은 바인딩을 의미합니다. 따라서 set_info 메서드의 동작은 그림 6.5와 같이 메서드 인자인 name, email, addr이라는 변수가 가리키고 있던 값을 self.name, self.email, self.addr이 바인딩하는 것입니다.

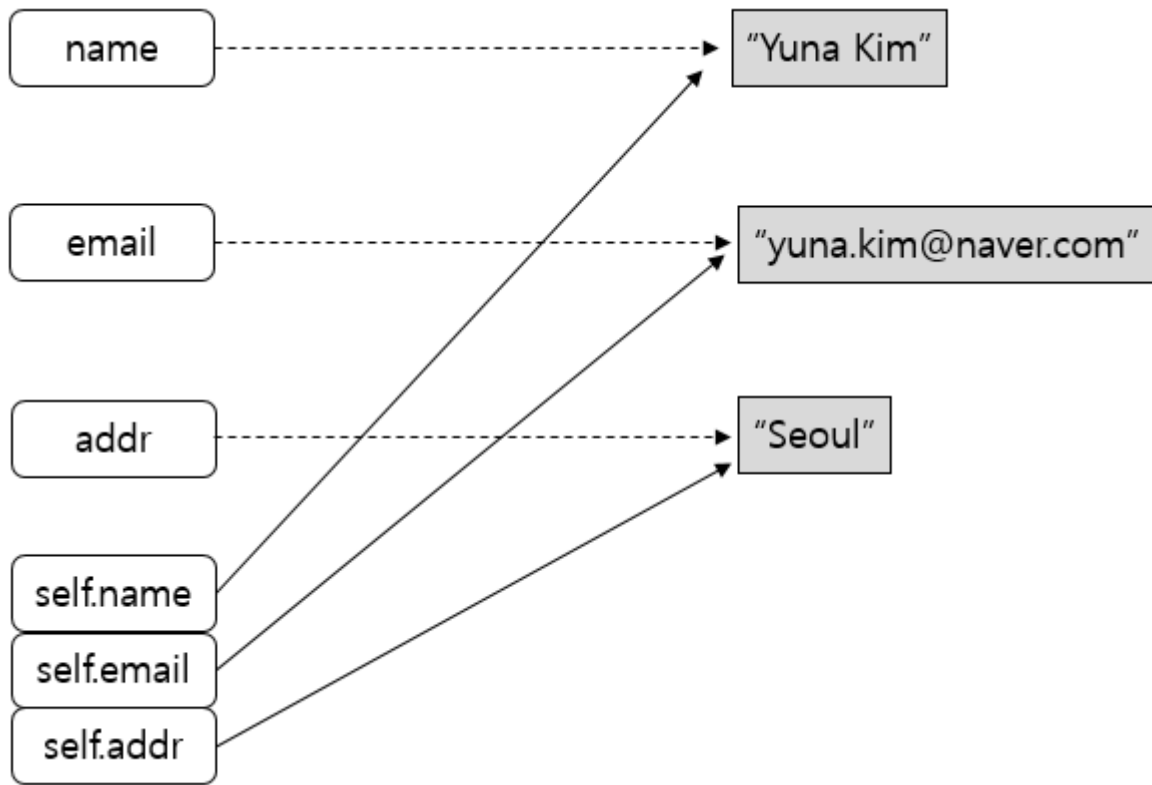


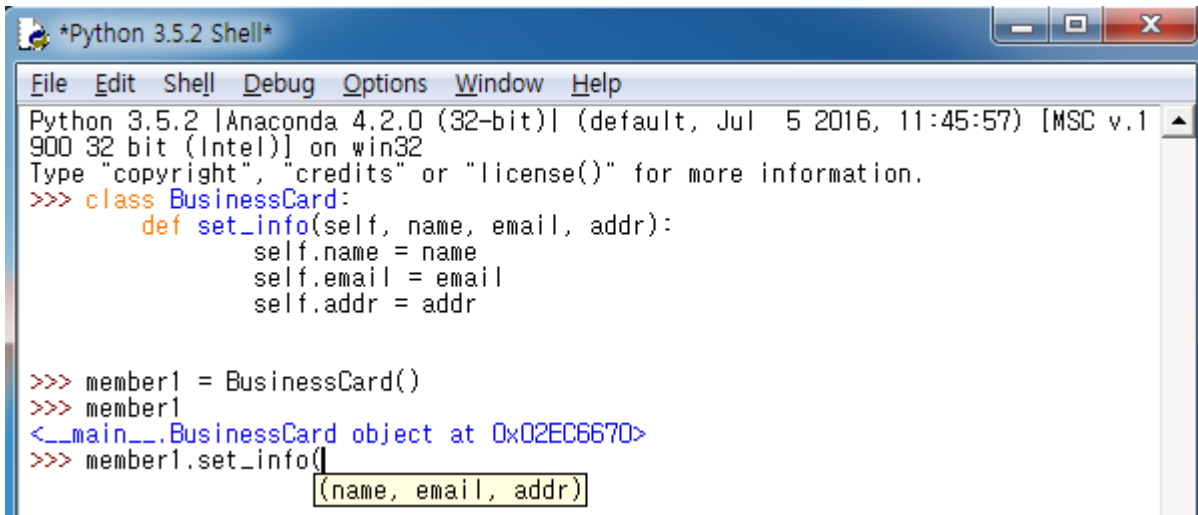
그림 6.5 변수의 바인딩

BusinessCard 클래스를 새롭게 정의했으므로 새롭게 정의된 클래스로 인스턴스를 생성해 봅시다. 붕어빵 이야기로 비유해 보면 붕어빵을 굽는 틀을 새롭게 바꿨으므로 새로 붕어빵을 구워보는 것입니다.

```
>>> member1 = BusinessCard()
>>> member1
<__main__.BusinessCard object at 0x030248F0>
>>>
```

새롭게 정의된 BusinessCard 클래스는 set_info 라는 메서드를 포함하고 있습니다. 따라서 member1 인스턴스는 set_info 메서드를 호출할 수 있습니다.

그림 6.6 을 보면 member1 이라는 클래스 인스턴스를 통해 set_info 라는 메서드를 호출할 수 있음을 확인할 수 있습니다. 단, 메서드에 인자를 전달하기 위해 파이썬 IDLE 에서 괄호를 입력하면 메서드의 인자가 네 개가 아니라 세 개임을 확인할 수 있습니다. 앞서 set_info 메서드를 정의할 때는 self, name, email, addr 의 네 개의 인자가 사용됐는데 메서드를 호출할 때는 왜 세 개만 사용될까요?



```
*Python 3.5.2 Shell*
File Edit Shell Debug Options Window Help
Python 3.5.2 [Anaconda 4.2.0 (32-bit)] (default, Jul 5 2016, 11:45:57) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr

>>> member1 = BusinessCard()
>>> member1
<__main__.BusinessCard object at 0x02EC6670>
>>> member1.set_info(
    (name, email, addr)
```

그림 6.6 클래스 인스턴스를 통한 메서드 호출

일단 다음 코드처럼 파이썬 IDLE 가 알려주는 대로 세 개의 인자만 `set_info` 메서드의 입력으로 전달합니다. 항상 그렇듯이 파이썬 IDLE 에서 에러가 발생하지 않았다면 정상적으로 코드가 실행된 것을 의미합니다.

```
>>> member1.set_info("Yuna Kim", "yunakim@naver.com", "Seoul")
```

```
>>>
```

`set_info` 메서드는 메서드 인자로 넘어온 값을 `self.name`, `self.email`, `self.addr` 로 바인딩했습니다. 그런데 현재 사용 가능한 변수는 클래스 인스턴스인 `member1` 뿐입니다. 어떻게 하면 `member1` 을 통해 `self.name`, `self.email`, `self.addr` 에 접근할 수 있을까요?

`self.name`, `self.email`, `self.addr` 과 같이 'self.변수이름'과 같은 형태를 갖는 변수를 인스턴스 변수라고 합니다. 인스턴스 변수는 클래스 인스턴스 내부의 변수를 의미합니다. 위 코드에서 `member1` 이라는 인스턴스를 생성한 후 `set_info` 메서드를 호출하면 메서드의 인자로 전달된 값을 인스턴스 내부 변수인 `self.name`, `self.email`, `self.addr` 이 바인딩하는 것입니다. 클래스를 정의하는 순간에는 여러분이 생성할 인스턴스의 이름이 `member1` 인지 모르기 때문에 `self` 라는 단어를 대신 사용하는 것입니다.

정리해보면 `set_info` 메서드 내에서 `self.name` 이라는 표현은 나중에 생성될 클래스 인스턴스 내의 `name` 변수를 의미합니다. 따라서 인스턴스 변수는 다음과 같이 인스턴스 이름을 적고 '.'를 붙인 후 인스턴스 변수의 이름을 지정하는 식으로 접근할 수 있습니다.

```
>>> member1.name
```

```
'Yuna Kim'
```

```
>>> member1.email
```

```
'yunakim@naver.com'
```

```
>>> member1.addr
```

```
'Seoul'
```

```
>>>
```

클래스는 한번 정의해두면 해당 클래스를 이용해 여러 클래스 인스턴스를 만들 수 있습니다. 아직도 개념이 잘 이해되지 않는다면 항상 붕어빵 틀을 생각하세요. 붕어빵 틀을 한 번 만들어두면 붕어빵을 계속해서 구울 수 있는 것과 비슷합니다.

이번에는 두 번째 회원에 대한 정보를 저장하는 인스턴스를 만들어 보겠습니다. 다음 코드를 보면 먼저 member2 라는 인스턴스를 생성한 후 set_info 메서드를 호출해 인스턴스에 값을 저장하는 것을 확인할 수 있습니다.

```
>>> member2 = BusinessCard()
```

```
>>> member2.set_info("Sarang Lee", "sarang.lee@naver.com", "Kyunggi")
```

```
>>>
```

새로 생성한 인스턴스 역시 member2 라는 이름을 통해 해당 인스턴스 내의 변수에 접근할 수 있습니다.

```
>>> member2.name
```

```
'Sarang Lee'
```

```
>>> member2.email
```

```
'sarang.lee@naver.com'
```

```
>>> member2.addr
```

```
'Kyunggi'
```

```
>>>
```

그림 6.7 은 클래스 인스턴스인 member1 과 member2 를 그림으로 표현해 본 것입니다. member1 과 member2 는 서로 동일한 이름의 인스턴스 변수 name, email, addr 을 갖고 있지만 각각 다른 데이터를 바인딩하고 있습니다.

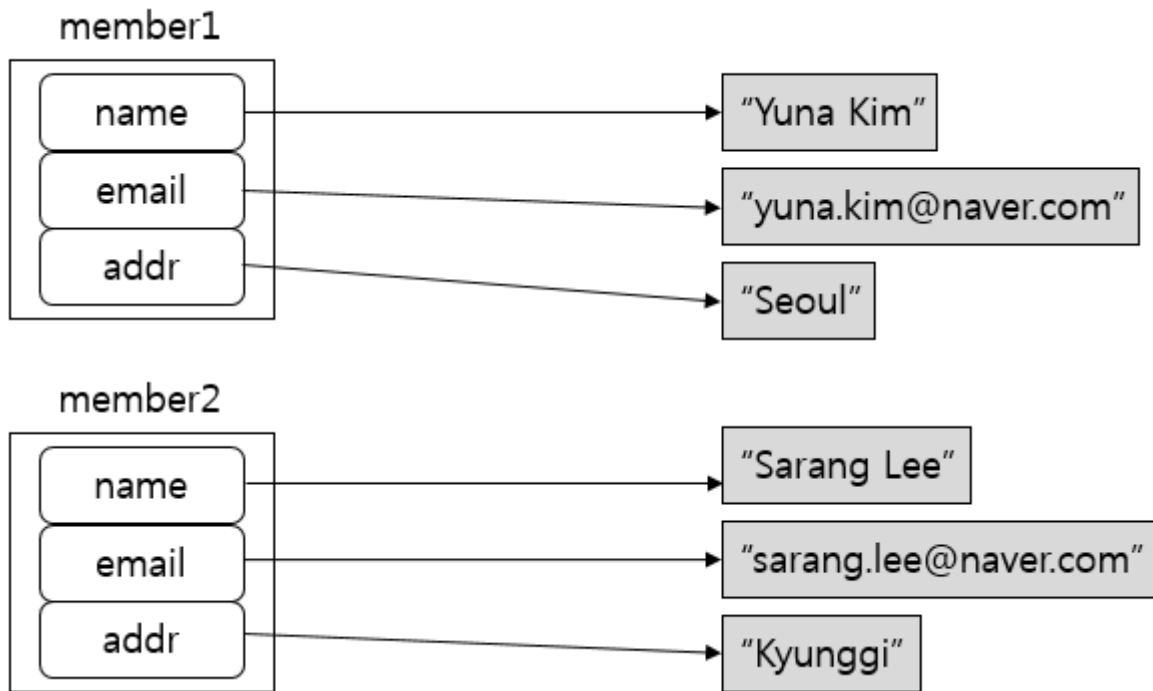


그림 6.7 클래스 인스턴스 상태

이번에는 BusinessCard 클래스에 명함을 출력하는 메서드를 추가해 보겠습니다. 다음과 같이 print_info 메서드를 추가해 BusinessCard 클래스를 새롭게 정의하기 바랍니다. 새로 추가된 print_info 메서드를 살펴보면 self 라는 인자 값만을 사용합니다. 앞서 간단히 설명한 것처럼 파이썬 클래스의 메서드는 항상 self 라는 기본 인자를 하나 갖고 있어야 하므로 self 를 제외하면 print_info 메서드는 사실 인자가 없는 함수나 마찬가지입니다.

```
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr
    def print_info(self):
        print("-----")
        print("Name: ", self.name)
        print("E-mail: ", self.email)
        print("Address: ", self.addr)
```

```
print("-----")
```

```
>>>
```

print_info 메서드의 내부를 살펴보면 print 함수를 통해서 self.name, self.email, self.addr 값을 출력하는 것을 확인할 수 있습니다. 앞서 set_info 메서드에서 이름을 저장할 때 self.name 에 저장했으므로 print_info 메서드에서도 self.name 을 통해 바로 데이터에 접근하면 됩니다.

BusinessCard 클래스를 재정의했으므로 다음과 같이 새로 인스턴스를 생성해 봅시다.

```
>>> member1 = BusinessCard()
```

```
>>> member1.set_info("YunaKim", "yuna.kim@naver.com", "Seoul")
```

```
>>>
```

member1 이라는 인스턴스에 값이 제대로 저장됐는지 확인하기 위해 member1.name 과 같이 인스턴스 변수에 바로 접근할 수도 있습니다. 하지만 BusinessCard 클래스에 print_info 메서드를 정의해뒀기 때문에 다음과 같이 해당 메서드의 호출을 통해 값을 화면에 출력해 볼 수 있습니다.

```
>>> member1.print_info()
```

```
-----
```

```
Name: YunaKim
```

```
E-mail: yuna.kim@naver.com
```

```
Address: Seoul
```

```
-----
```

```
>>>
```

2) 클래스 생성자

6.1 절에서 파이썬의 클래스에 대해 배웠습니다. 지금까지 배운 내용을 정리해보면 다음과 같습니다.

- 파이썬의 클래스를 이용하면 프로그래머가 원하는 새로운 타입을 만들 수 있다.
- 생성된 타입은 데이터와 데이터를 처리하는 메서드(함수)로 구성돼 있다.

그럼 지난 시간에 만든 클래스를 다시 한 번 살펴볼까요?

```
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr
    def print_info(self):
        print("-----")
        print("Name: ", self.name)
        print("E-mail: ", self.email)
        print("Address: ", self.addr)
        print("-----")
```

```
>>>
```

파이썬의 class 키워드를 통해 BusinessCard 라는 새로운 타입을 만들었으니 인스턴스를 생성해야겠죠? 다시 한번 말씀드리면 클래스 인스턴스를 생성하려면 '클래스 이름()'과 같이 적으면 됩니다. 그리고 생성된 인스턴스에 점(.)을 찍어서 인스턴스 변수나 메서드에 접근할 수 있었습니다.

```
>>> member1 = BusinessCard()
>>> member1.set_info("kim", "kim@gmail.com", "USA")
```



```
>>>
```

위 코드를 살펴보면 먼저 인스턴스를 생성하고 생성된 인스턴스에 데이터를 입력하는 순으로 코드가 구성돼 있습니다. 붕어빵으로 비유해 보면 붕어빵 틀(클래스)을 이용해 팔소를 넣지 않은 상태로 붕어빵을 구운 후(인스턴스 생성) 나중에 다시 붕어빵 안으로 팔소를 넣는 것과 비슷합니다. 어떻게 하면 클래스 인스턴스 생성과 초깃값 입력을 한 번에 처리할 수 있을까요?

파이썬 클래스에는 인스턴스가 생성과 동시에 자동으로 호출되는 메서드인 생성자가 존재합니다. 참고로 생성자는 C++나 자바 같은 객체 지향 프로그래밍 언어에도 있는 개념입니다. 파이썬에서는 `__init__(self)`와 같은 이름의 메서드를 생성자라고 하며, 파이썬 클래스에서 `__`로 시작하는 함수는 모두 특별한 메서드를 의미합니다.

다음은 생성자인 `__init__(self)` 메서드를 가진 MyClass 클래스를 정의한 것입니다. 앞서 설명한 것처럼 생성자의 첫 번째 인자도 항상 self 이어야 합니다. 생성자 내부에는 print 문을 통해 간단한 메시지를 출력했습니다.

```
>>> class MyClass:
    def __init__(self):
        print("객체가 생성되었습니다.")
```

```
>>>
```

MyClass 라는 클래스를 정의했으니 이제 이 클래스의 인스턴스를 생성해 보겠습니다. 다음 코드를보면 인스턴스를 생성하자마자 화면에 메시지가 출력되는 것을 확인할 수 있습니다. 이는 인스턴스가 생성되는 시점에 자동으로 생성자인 `__init__(self)` 메서드가 호출됐기 때문입니다. 참고로 init 이라는 영어 단어는 '초기화하다'라는 뜻이 있는 initialize 의 약어입니다.

```
>>> inst1 = MyClass()

객체가 생성되었습니다.
```

```
>>>
```

클래스 생성자를 이해했다면 BusinessCard 클래스를 수정해 인스턴스의 생성과 동시에 명함에 필요한 정보를 입력받도록 클래스를 새롭게 정의해 봅시다. 눈치가 빠르신 분들은 벌써 정답을 맞혔지요? 기존에 구현했던 BusinessCard 클래스의 set_info 메서드가 데이터를 입력받는 역할을 수행했으므로 set_info 메서드를 `__init__` 메서드로 이름만 변경해주면 되겠습니다.

```
>>> class BusinessCard:
    def __init__(self, name, email, addr):
        self.name = name
```

```

        self.email = email

        self.addr = addr

    def print_info(self):

        print("-----")

        print("Name: ", self.name)

        print("E-mail: ", self.email)

        print("Address: ", self.addr)

        print("-----")

```

```
>>>
```

새로 정의된 BusinessCard 클래스의 생성자는 인자가 4 개임을 확인할 수 있습니다. 물론 첫 번째 인자인 self 는 생성되는 인스턴스를 의미하고 자동으로 값이 전달되므로 인스턴스를 생성할 때 명시적으로 인자를 전달해야 하는 것은 3 개입니다. 따라서 인스턴스를 생성할 때 3 개의 인자를 전달해주지 않으면 오류가 발생합니다. 생성자 호출단계에서 오류가 발생하면 인스턴스도 정상적으로 생성되지 않게 됩니다.

```
>>> member1 = BusinessCard()
```

Traceback (most recent **call last**):

File "<pyshell#12>", line 1, in <module>

```
member1 = BusinessCard()
```

TypeError: __init__() missing 3 required positional arguments: 'name', 'email', and 'addr'

```
>>>
```

새로 정의된 BusinessCard 클래스는 생성자에서 3 개의 인자(name, email, addr)를 받기 때문에 다음과 같이 인스턴스를 생성할 때 3 개의 인자를 전달해야 정상적으로 인스턴스가 생성됩니다. member1 이라는 인스턴스가 생성된 후에는 인스턴스 메서드를 호출해 인스턴스 변수 값을 화면에 출력할 수 있습니다. 어떨까요? 클래스의 생성자를 사용하니 인스턴스의 생성과 초깃값 저장을 한 번에 할 수 있어 편리하지요?

```
>>> member1 = BusinessCard("Kangsan Lee", "kangsan.lee", "USA")
```

```
>>> member1.print_info()
```

```
-----
```

```
Name:  Kangsan Lee
```

```
E-mail:  kangsan.lee
```

```
Address:  USA
```

```
-----
```

```
>>>
```

3) self 이해하기

클래스 내에 정의된 함수를 메서드라고 부른다고 했습니다. 그리고 메서드의 첫 번째 인자는 항상 self 여야 한다고 했습니다. 하지만 메서드의 첫 번째 인자가 항상 self 여야 한다는 것은 사실 틀린 말입니다. 이번 절에서는 파이썬 클래스에서 self의 정체를 확실히 이해해 봅시다.

먼저 다음과 같이 두 개의 메서드가 정의된 Foo 클래스를 만들어 봅시다. 여기서 눈여겨보아야 할 점은 func1() 메서드의 첫 번째 인자가 self가 아님에도 클래스를 정의할 때 에러가 발생하지 않는다는 점입니다.

```
>>> class Foo:
    def func1():
        print("function 1")
    def func2(self):
        print("function 2")
```

```
>>>
```

일단 클래스를 정의했으니 해당 클래스에 대한 인스턴스를 생성해보겠습니다. 그리고 생성된 인스턴스를 통해 인스턴스 메서드를 호출해보겠습니다. Foo 클래스의 func2 메서드는 메서드의 인자가 self 뿐이므로 실제 메서드를 호출할 때는 인자를 전달할 필요가 없습니다.

```
>>> f = Foo()
```

```
>>> f.func2()
```

```
function 2
```

```
>>>
```

위 코드에서 메서드 호출의 결과를 보면 화면에 정상적으로 'function 2'가 출력된 것을 볼 수 있습니다. 참고로 func2 메서드의 첫 번째 인자는 self지만 호출할 때는 아무것도 전달하지 않는 이유는 첫 번째 인자인 self에 대한 값은 파이썬이 자동으로 넘겨주기 때문입니다.

그렇다면 func1 메서드처럼 메서드를 정의할 때부터 아무 인자도 없는 경우에는 어떻게 될까요? 다음과 같이 인스턴스를 통해 func1()을 호출해보면 오류가 발생합니다. 오류 메시지를 살펴보면 "func1()은 인자가 없지만 하나를 받았다"라는 것을 볼 수 있습니다. 이는 앞서 저자가 설명한 것처럼 파이썬 메서드의 첫 번째 인자로 항상 인스턴스가 전달되기 때문에 발생하는 문제입니다.

```
>>> f.func1()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#25>", line 1, in <module>
```

```
f.func1()
```

```
TypeError: func1() takes 0 positional arguments but 1 was given
```

```
>>>
```

이번에는 self의 정체를 좀 더 확실히 밝혀보기 위해 파이썬 내장 함수인 id()를 이용해 인스턴스가 메모리에 할당된 주솟값을 확인해보겠습니다. 다음 코드처럼 Foo 클래스를 새로 정의합니다. func2 메서드가 호출될 때 메서드의 인자로 전달되는 self의 id 값을 화면에 출력하는 기능이 추가되었습니다.

```
>>> class Foo:
```

```
    def func1():
```

```
        print("function 1")
```

```
    def func2(self):
```

```
        print(id(self))
```

```
        print("function 2")
```

```
>>>
```

Foo 클래스를 새롭게 정의했으므로 인스턴스를 다시 만든 후 id() 내장함수를 이용해 생성된 인스턴스가 할당된 메모리 주소를 확인해 봅시다.

```
>>> f = Foo()
```

```
>>> id(f)
```

```
43219856
```

```
>>>
```

생성된 인스턴스가 메모리의 43219856 번지에 있음을 확인할 수 있습니다. 참고로 이 값은 해당 코드의 실행 환경에 영향을 받게 되므로 여러분이 직접 실행했을 때 이 값과 다른 값이 나올 수 있습니다.

위 코드에서 f와 생성된 인스턴스의 관계를 그림으로 나타내면 그림 6.8 과 같습니다. Foo 클래스에 대한 인스턴스는 메모리의 43219856 번지부터 할당돼 있고 변수 f는 인스턴스의 주솟값을 담고 있습니다. 일단 인스턴스가 할당된 메모리 주솟값을 기억해두기 바랍니다. 곧 놀라운 광경을 목격할 수 있을 것입니다.

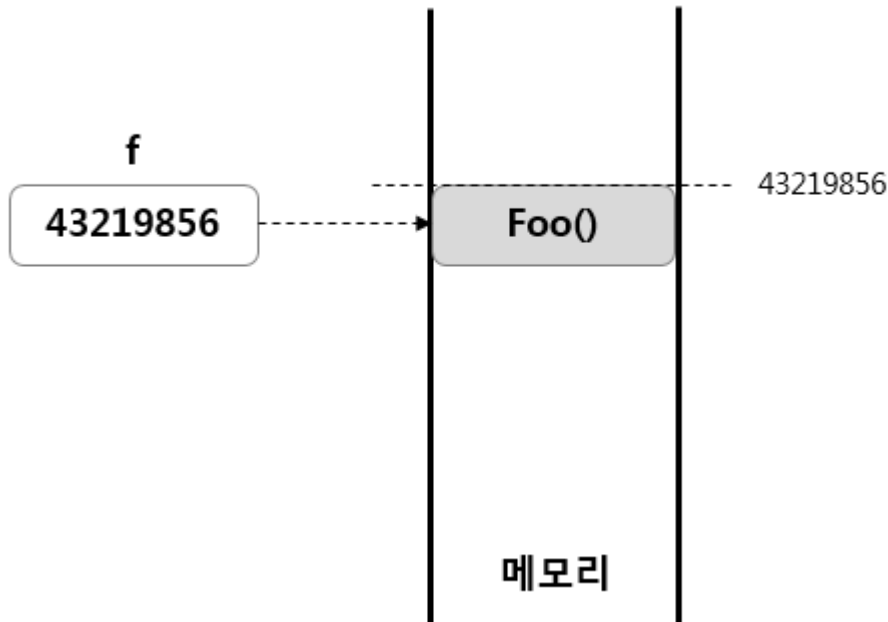


그림 6.8 인스턴스의 메모리 주소 확인

이번에는 인스턴스 f를 이용해 func2 메서드를 호출해보기 바랍니다. 다음 코드를 살펴보면 func2 메서드를 호출할 때 아무런 값도 전달하지 않았습니다.

```
>>> f.func2()
43219856
function 2
>>>
```

실행 결과를 살펴보면 43219856 이라는 값이 출력됨을 확인할 수 있습니다. Foo 클래스를 정의할 때 id(self)를 출력하게 했는데 id(self)의 값이 바로 43219856 인 것입니다. 이 값은 그림 6.8 에서 볼 수 있듯이 f라는 변수가 바인딩하고 있는 인스턴스의 주솟값과 동일합니다. 즉, 클래스 내에 정의된 self는 클래스 인스턴스임을 알 수 있습니다.

아직 이 부분이 잘 이해되지 않는다면 객체를 하나 더 만들어 보겠습니다. 새로 생성한 객체는 f2가 가리키고 있는데, id()를 통해 확인해보니 주소가 47789232입니다. f2를 통해 func2 메서드를 호출해보면 47789232가 출력됐습니다. 이 값은 바로 f2가 가리키고 있는 객체를 의미합니다.

```
>>> f2 = Foo()
>>> id(f2)
47789232
>>> f2.func2()
47789232
function 2
>>>
```

파이썬의 클래스는 그 자체가 하나의 네임스페이스이기 때문에 인스턴스 생성과 상관없이 클래스 내의 메서드를 직접 호출할 수 있습니다.

```
>>> Foo.func1()
function 1
>>>
```

위 코드는 func1() 메서드를 호출했지만 앞서 인스턴스를 통해 메서드를 호출했던 것과는 달리 오류가 발생하지 않는 것을 확인할 수 있습니다. 왜냐하면 인스턴스.메서드() 형태로 호출한 것과 달리 이번에는 클래스이름.메서드() 형태로 호출했기 때문입니다. 그렇다면 func2()에 대해서도 클래스 이름을 통해 호출해봅시다. 그림 6.9 와 같이 클래스 이름을 통해 func2() 메서드를 호출하려고 하면 self 위치에 인자를 전달해야 한다고 파이썬 인터프리터가 알려줍니다.

```
>>> Foo.func2(
      (self)
Ln: 68 Col: 14
```

그림 6.9 클래스 이름을 이용한 메서드 호출

self 위치에 인자를 전달하지 않고 메서드를 호출하면 그림 6.10 과 같이 오류가 발생합니다. 오류 메시지를 확인하면 func2()를 호출할 때 인자를 하나 빠트렸음을 알 수 있습니다. 즉, 인자를 하나 전달해야 하는데 전달하지 않아서 오류가 발생한 것입니다.

```
>>> Foo.func2()
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    Foo.func2()
TypeError: func2() missing 1 required positional argument: 'self'
>>> |
Ln: 40 Col: 4
```

그림 6.10 클래스 이름을 이용한 메서드 호출 예러

그럼 도대체 어떤 값을 전달하면 되는 걸까요? 앞에서 메서드의 self 로 전달되는 것은 인스턴스 자체라고 설명했습니다. 따라서 클래스에 대한 인스턴스를 생성한 후 해당 인스턴스를 전달하면 됩니다. 다음과 같이 새로운 객체를 만들고 이를 f3 이 가리키게 합니다. id(f3) 구문 호출을 통해 f3 이 가리키는 객체의 주소가 47789136 임을 확인할 수 있습니다.

```
>>> f3 = Foo()
```

```
>>> id(f3)
```

```
47789136
```

```
>>>
```

다시 한 번 클래스 이름인 Foo 를 이용해 func2 메서드를 호출해보겠습니다. 앞서 func2 메서드는 인자를 하나 필요로 하며, 해당 인자는 인스턴스여야 한다고 말씀드렸습니다. 현재 f3 은 새로 생성한 인스턴스를 바인딩하고 있으므로 func2 메서드의 인자로 f3 을 전달해주면 됩니다.

```
>>> Foo.func2(f3)
```

```
47789136
```

```
func2
```

```
>>>
```

그렇다면 인스턴스를 통해 func2 를 호출하는 것과 클래스 이름을 통해 func2 를 호출하는 것은 어떤 차이가 있을까요? 결론부터 말씀드리면 둘 사이에는 아무런 차이가 없습니다. 다만 '인스턴스.메서드()'냐 '클래스.메서드(인스턴스)'냐 라는 차이가 있을 뿐입니다. 보통은 '인스턴스.메서드()'와 같은 방식을 주로 사용합니다.

```
>>> f3.func2()
```

```
47789136
```

```
func2
```

```
>>>
```


4) 클래스 네임스페이스

클래스와 인스턴스의 차이를 정확히 이해하는 것은 매우 중요합니다. 이를 위해서는 먼저 네임스페이스라는 개념을 알아야 합니다. 네임스페이스라는 것은 변수가 객체를 바인딩할 때 그 둘 사이의 관계를 저장하고 있는 공간을 의미합니다. 예를 들어, 'a = 2'라고 했을 때 a라는 변수가 2라는 객체가 저장된 주소를 가지고 있는데 그러한 연결 관계가 저장된 공간이 바로 네임스페이스입니다.

파이썬의 클래스는 새로운 타입(객체)을 정의하기 위해 사용되며, 모듈과 마찬가지로 하나의 네임스페이스를 가집니다. 먼저 Stock 클래스를 정의해봅시다.

```
>>> class Stock:
    market = "kospì"
```

```
>>>
```

파이썬 IDLE 에서 dir() 내장 함수를 호출해보면 리스트로 된 반환값을 확인할 수 있습니다. 여기서 두 개의 언더바로 시작하는 것은 파이썬에서 이미 사용 중인 특별한 것들입니다. 이를 제외하고 보면 조금 전에 정의했던 Stock 클래스의 이름이 포함된 것을 확인할 수 있습니다.

```
>>> dir()
['Stock', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
```

```
>>>
```

dir() 내장함수의 결과값에 Stock 클래스가 들어있기 때문에 앞으로는 프롬프트에 Stock 을 입력해도 오류가 발생하지 않습니다. 그러나 Stock1 이라는 이름은 존재하지 않기 때문에 입력하면 오류가 발생합니다.

```
>>> Stock
```

```
<class '__main__.Stock'>
```

```
>>> Stock1
```

```
Traceback (most recent call last):
```

```
File "<pyshell#5>", line 1, in <module>
```

```
Stock1
```

```
NameError: name 'Stock1' is not defined
```

```
>>>
```

파이썬에서는 클래스가 정의되면 그림 6.11 과 같이 하나의 독립적인 네임스페이스가 생성됩니다. 그리고 클래스 내에 정의된 변수나 메서드는 그 네임스페이스 안에 파이썬 딕셔너리 타입으로 저장됩니다. Stock 클래스는 그림 6.11 과 같이 Stock 이라는 네임스페이스 안에 'market':'kospì'라는 값을 가진 딕셔너리를 포함합니다.



그림 6.11 파이썬 클래스 네임스페이스

Stock 클래스의 네임스페이스를 파이썬 코드로 확인하려면 클래스의 `__dict__` 속성을 확인하면 됩니다. 딕셔너리 타입에 'market':'kospì'라는 키와 값 쌍이 존재하는 것을 확인할 수 있습니다.

```
>>> Stock.__dict__
```

```
mappingproxy({'market': 'kospì', '__module__': '__main__', '__dict__': <attribute '__dict__' of  
'Stock' objects>, '__doc__': None, '__weakref__': <attribute '__weakref__' of 'Stock' objects>})
```

```
>>>
```

클래스가 독립적인 네임스페이스를 가지고 클래스 내의 변수나 메서드를 네임스페이스에 저장하고 있으므로 다음과 같이 클래스 내의 변수에 접근할 수 있는 것입니다.

```
>>> Stock.market
```

```
'kospì'
```

```
>>>
```

이번에는 인스턴스를 생성해보겠습니다. 다음과 같이 서로 다른 두 개의 인스턴스를 생성해보기 바랍니다. 생성된 인스턴스에 대한 id 값을 확인해보면 두 인스턴스가 서로 다른 메모리에 위치하는 것을 확인할 수 있습니다.

```
>>> s1 = Stock()
```

```
>>> s2 = Stock()
```

```
>>> id(s1)
```

```
50572464
```

```
>>> id(s2)
```

```
50348240
```

```
>>>
```

파이썬은 인스턴스를 생성하면 인스턴스별로 별도의 네임스페이스를 유지합니다. 즉, 위의 코드를 그림으로 표현하면 그림 6.12 와 같습니다.

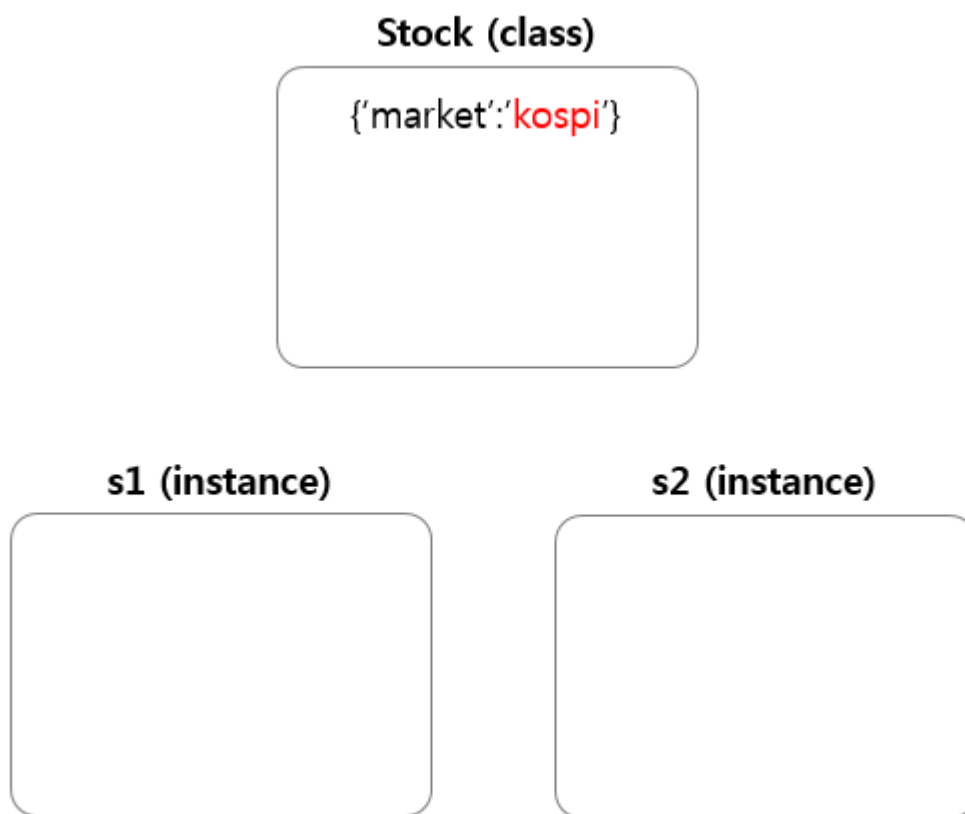


그림 6.12 클래스와 인스턴스의 네임스페이스

먼저 생성된 s1, s2 인스턴스가 네임스페이스에 있는지 코드를 통해 확인해 봅시다. `dir()` 내장함수의 반환값을 확인하면 s1, s2 가 Stock 과 마찬가지로 존재하는 것을 확인할 수 있습니다.

```
>>> dir()
```

```
['Stock', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 's1', 's2']
```

```
>>>
```

생성된 s1, s2 인스턴스 각각에 대한 네임스페이스도 확인해봅시다. 클래스 또는 인스턴스에 대한 네임스페이스를 확인하려면 `__dict__` 속성을 확인하면 됩니다.

```
>>> s1.__dict__
```

```
{}
```

```
>>> s2.__dict__
```

```
{}
```

```
>>>
```

위의 코드를 보면 s1 과 s2 인스턴스의 네임스페이스는 현재 비어 있는 것을 확인할 수 있습니다. s1 인스턴스에 market 이라는 변수를 추가해 봅시다. 그런 다음 다시 `__dict__` 속성을 확인해보면 'market': 'kosdaq'이라는 키:값 쌍이 추가된 것을 볼 수 있습니다.

```
>>> s1.market = 'kosdaq'
```

```
>>> s1.__dict__
```

```
{'market': 'kosdaq'}
```

```
>>>
```

그러나 여전히 s2 인스턴스의 네임스페이스는 비어 있는 상태입니다. 현재 Stock 클래스와 s1, s2 인스턴스의 네임스페이스를 그림으로 나타내면 그림 6.13 과 같습니다.

```
>>> s2.__dict__
```

```
{}
```

```
>>>
```

Stock (class object)

{'market': 'kospi'}

s1 (instance)

{'market': 'kosdaq'}

s2 (instance)

그림 6.13 클래스와 인스턴스의 네임스페이스 상태

현재 Stock 클래스로부터 s1, s2 라는 두 개의 인스턴스를 생성했습니다. s1 인스턴스는 market 이라는 변수를 가지고 있지만 s2 의 네임스페이스에는 변수나 메서드가 존재하지 않습니다. 만약 s1.market, s2.market 과 같이 인스턴스를 통해 market 이라는 값에 접근하면 어떻게 될까요?

```
>>> s1.market
```

```
'kosdaq'
```

```
>>> s2.market
```

```
'kospi'
```

```
>>>
```

위의 코드를 참조하면 s1.market 의 값은 'kosdaq'입니다. 이것은 그림 6.13 과 같이 s1 인스턴스의 네임스페이스에 'market': 'kosdaq'이라는 키:값 쌍이 존재하기 때문에 가능합니다. 그런데 s2 인스턴스의 반환값을 살펴보면 조금 이상합니다. 왜냐하면 현재 s2 의 네임스페이스(딕셔너리)에는 아무런 값도 존재하지 않기 때문입니다. s2 의 네임스페이스에는 변수나 메서드가 존재하지 않지만 s2.market 의 값으로 'kospi'라는 문자열이 반환되는 이유는 그림 6.14 와 같이 동작하기 때문입니다.

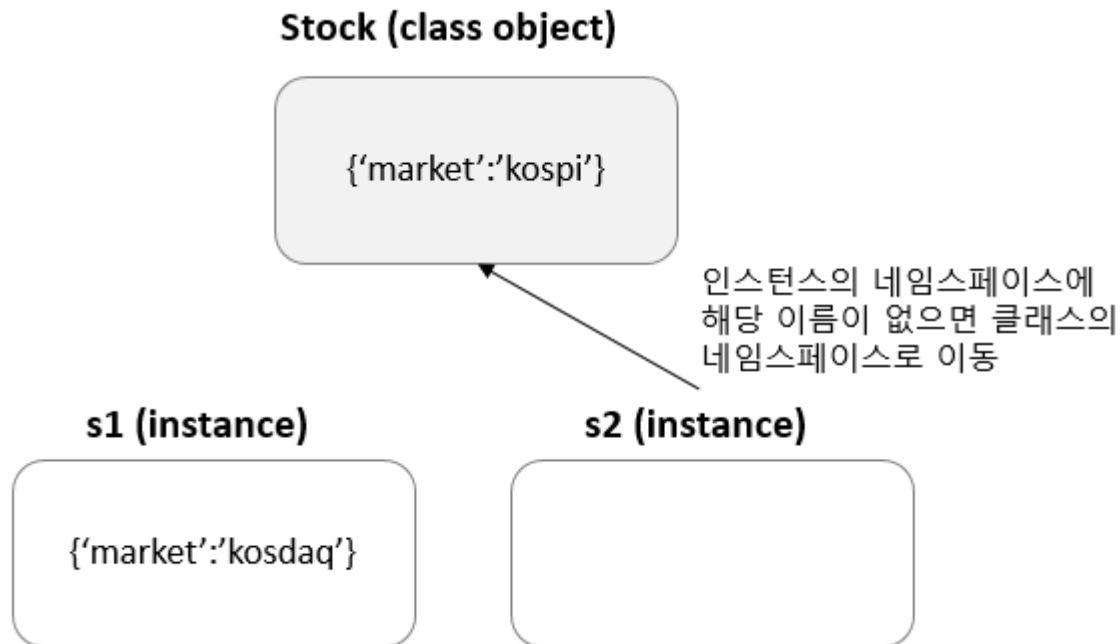


그림 6.14 클래스 및 인스턴스의 네임스페이스 참조 방식

s2 인스턴스를 통해 변수에 접근하면 파이썬은 먼저 s2 인스턴스의 네임스페이스에서 해당 변수가 존재하는지 찾습니다. s2의 네임스페이스에 해당 변수가 존재하지 않으면 s2 인스턴스의 클래스의 네임스페이스로 가서 다시 변수를 찾게 됩니다. 즉, s2.market이라는 문장이 실행되면 Stock 클래스의 네임스페이스에 있는 'market': 'kospi' 키:값 쌍에서 'kospi'라는 문자열을 출력하게 됩니다.

이번에는 인스턴스의 네임스페이스에도 없고 클래스의 네임스페이스에도 없는 변수에 접근해 봅시다. 이 경우 volume이라는 값이 s2 인스턴스의 네임스페이스에 없으므로 Stock 클래스에서 찾게 되는데, Stock 클래스의 네임스페이스에도 volume이라는 값이 없으므로 오류가 발생합니다.

```

>>> s2.volume
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    s2.volume
AttributeError: 'Stock' object has no attribute 'volume'

>>>
  
```

5) 클래스 변수와 인스턴스 변수

6.4 절에서는 클래스의 네임스페이스와 인스턴스의 네임스페이스, 그리고 그 둘 사이의 관계에 대해 배웠습니다. 이번 절에서는 초보자들이 많이 어려워하는 개념 중 하나인 클래스 변수(class variable)와 인스턴스 변수(instance variable)에 대해 살펴보겠습니다.

다음은 은행 계좌를 클래스로 표현한 것입니다. Account 클래스에는 생성자와 소멸자가 있습니다. 생성자 (`__init__`)가 클래스의 인스턴스가 생성될 때 자동으로 호출되는 함수라면 소멸자 (`__del__`)는 클래스의 인스턴스가 소멸될 때 자동으로 호출되는 함수입니다.

```
>>> class Account:
    num_accounts = 0
    def __init__(self, name):
        self.name = name
        Account.num_accounts += 1
    def __del__(self):
        Account.num_accounts -= 1
```

```
>>>
```

Account 클래스에는 `num_accounts` 와 `self.name` 이라는 두 종류의 변수가 존재합니다. `num_accounts` 와 같이 클래스 내부에 선언된 변수를 클래스 변수라고 하며, `self.name` 과 같이 `self` 가 붙어 있는 변수를 인스턴스 변수라고 합니다. 클래스 변수는 Account 클래스의 네임스페이스에 위치하며, `self.name` 과 같은 인스턴스 변수는 인스턴스의 네임스페이스에 위치하게 됩니다.

그렇다면 언제 클래스 변수를 사용해야 하고 언제 인스턴스 변수를 사용해야 할까요? 이에 대한 답은 간단한 코드를 작성해보면서 천천히 설명해 드리겠습니다. 여러분이 은행에 가서 계좌를 개설하면 새로운 계좌가 하나 개설됩니다. 이러한 상황을 파이썬으로 표현하면 다음과 같이 Account 클래스의 인스턴스를 생성하는 것에 해당합니다.

```
>>> kim = Account("kim")
```

```
>>> lee = Account("lee")
```

```
>>>
```

생성된 kim 과 lee 인스턴스에 계좌 소유자 정보가 제대로 저장돼 있는지 확인해 봅시다. 각 계좌에 대한 소유자 정보는 인스턴스 변수인 name 이 바인딩하고 있습니다.

```
>>> kim.name
'kim'
>>> lee.name
'lee'
>>>
```

그렇다면 지금까지 은행에서 개설된 계좌는 총 몇 개일까요? 네, 정답은 'kim'과 'lee'에게 하나씩 개설됐기 때문에 두 개겠죠? kim 인스턴스나 lee 인스턴스를 통해 num_accounts 라는 이름에 접근하면 총 계좌 개설 개수가 2 개로 나오는 것을 알 수 있습니다.

```
>>> kim.num_accounts
2
>>> lee.num_accounts
2
>>>
```

물론 지금까지 공부를 잘 해오신 분들은 kim.num_accounts 에서 먼저 인스턴스의 네임스페이스에서 num_accounts 를 찾았지만 해당 이름이 없어서 클래스의 네임스페이스로 이동한 후 다시 해당 이름을 찾았고 그 값이 반환된 것임을 아실 것입니다.

위와 같이 여러 인스턴스 간에 서로 공유해야 하는 값은 클래스 변수를 통해 바인딩해야 합니다. 왜냐하면 파이썬은 인스턴스의 네임스페이스에 없는 이름은 클래스의 네임스페이스에서 찾아보기 때문에 이러한 특성을 이용하면 클래스 변수가 모든 인스턴스에 공유될 수 있기 때문입니다. 참고로 클래스 변수에 접근할 때 아래와 같이 클래스 이름을 사용할 수도 있습니다.

```
>>> Account.num_accounts
2
>>>
```


지금까지 작성한 코드에서 클래스 변수와 인스턴스 변수를 그림으로 나타내면 그림 6.15 와 같습니다. 앞으로 클래스 변수와 인스턴스 변수가 헷갈릴 때마다 이 그림을 기억하기 바랍니다.

Account (class object)

{'num_accounts':0}

kim (instance)

{'name':'kim'}

lee (instance)

{'name':'lee'}

그림 6.15 클래스 변수와 인스턴스 변수

6) 클래스 상속

상속이란 사람이 사망함에 따라 사망자의 재산 및 신분상의 지위에 대한 포괄적인 승계를 의미합니다. 드라마에서 보면 부모님으로부터 많은 재산을 상속받은 사람들을 종종 볼 수 있지요? 상속하는 사람 입장은 잘 모르겠지만 상속자들은 분명 상속받지 않은 경우보다 대부분 좋은 경우가 많을 것입니다. 특히 많은 재산을 상속받는 경우라면 더욱 좋겠지요?

저자가 프로그래밍 책에서 갑자기 상속 이야기를 한 이유는 객체 지향 프로그래밍을 지원하는 프로그래밍 언어는 클래스에서 상속 기능을 지원하기 때문입니다. 자식이 부모님으로부터 재산 등을 상속받는 것처럼 다른 클래스에 이미 구현된 메서드나 속성을 상속한 클래스에서는 그러한 메서드나 속성을 그대로 사용할 수 있게 됩니다.

클래스의 상속을 또 다른 관점에서 생각해보면 클래스를 상속한다는 것은 부모 클래스의 능력을 그대로 전달받는 것을 의미합니다. 인간으로 치면 부모로부터 유전형질을 물려받아 부모의 능력을 그대로 물려받는 것과 비슷합니다.

일단 노래를 잘 부르는 부모 클래스가 있다고 생각해 봅시다. 이를 파이썬으로 표현하면 다음과 같이 노래를 부르는 메서드가 포함된 클래스를 정의할 수 있습니다.

```
>>> class Parent:
    def can_sing(self):
        print("Sing a song")
```

```
>>>
```

Parent 클래스를 정의했으니 클래스의 인스턴스를 생성해 보겠습니다. 그리고 노래를 정말 할 수 있는지 메서드를 호출해 확인해 보겠습니다.

```
>>> father = Parent()
>>> father.can_sing()
```

```
Sing a song
```

```
>>>
```

이번에는 노래를 잘 부르는 Parent 클래스로부터 상속받은 운이 좋은 자식 클래스를 정의해 봅시다. 클래스의 이름은 LuckyChild 라고 하겠습니다. 클래스를 정의할 때 다른 클래스로부터 상속받고자 한다면 새로 정의할 클래스 이름 다음에 괄호를 사용해 상속받고자 하는 클래스의 이름을 지정하면 됩니다.

LuckyChild 클래스는 상속받기 전까지는 아무런 능력이 없어서 내부에 메서드를 구현하지 않고 pass 만 적어 주었습니다.

```
>>> class LuckyChild(Parent):
```

```
    pass
```

```
>>>
```

Parent 클래스로부터 상속받은 LuckyChild 클래스에 대한 인스턴스를 생성한 후 노래를 시켜보겠습니다. 중요한 점은 현재 LuckyChild 클래스에는 어떤 메서드도 존재하지 않는다는 것입니다. 다음 코드를 보면 역시나 부모 클래스로부터 상속받아서 그런지 자신은 메서드를 포함하고 있지 않지만 바로 노래를 부를 수 있군요.

```
>>> child1 = LuckyChild()
```

```
>>> child1.can_sing()
```

```
Sing a song
```

```
>>>
```

이번에는 부모로부터 어떤 능력이나 재산도 상속받지 못한 운이 좋지 않은 자식 클래스(UnLuckyChild)를 만들어 보겠습니다. 인간 세상으로 치면 평범한 서민 클래스라고 볼 수 있겠습니다.

```
>>> class UnLuckyChild:
```

```
    pass
```

```
>>>
```

운이 좋지 않은 UnLuckyChild 클래스에 대한 인스턴스를 생성한 후 노래를 시켜봅시다. 역시나 상속도 받지 못했고 자신도 아무런 메서드가 없으므로 노래를 부를 수 없습니다. 이처럼 능력이나 재산을 상속받지 못했다면 자신이 직접 노래를 잘할 수 있도록 연습해야겠지요? 프로그래밍 측면에서 보면 직접 메서드를 구현해야 한다는 뜻입니다.

```
>>> child2 = UnLuckyChild()
```

```
>>> child2.can_sing()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#53>", line 1, in <module>
```

```
child2.can_sing()
```

```
AttributeError: 'UnLuckyChild' object has no attribute 'can_sing'
```

```
>>>
```

인간 세계에서는 상속을 받으면 좋겠구나, 라고 생각되겠지만 프로그래밍할 때 왜 상속을 해야 하는지 여전히 이해되지 않는 분도 계실 겁니다. 객체 지향 프로그래밍에서는 어떤 클래스를 상속하면 부모 클래스의 모든 것을 내 것처럼 사용할 수 있고, 자신의 클래스에 메서드를 더 구현한다면 '플러스알파'가 되는 것입니다. 즉, 부모 클래스의 능력을 밑바탕으로 깔고 거기서부터 한 번 더 업그레이드되는 것입니다.

그럼 이번에는 노래도 잘 부르고 춤도 잘 추는 운이 좋은 자식 클래스 2(LuckyChild2)를 정의해 보겠습니다.

```
>>> class LuckyChild2(Parent):
```

```
    def can_dance(self):
```

```
        print("Shuffle Dance")
```

```
>>>
```

LuckyChild2 클래스에 대한 인스턴스를 생성한 후 노래와 춤을 시켜봅시다. LuckyChild2 클래스는 부모로부터 노래 부르는 능력을 상속받았고 자기 자신이 춤추는 능력을 갖추고 있어 노래도 부르고 춤도 출 수 있게 되었습니다.

```
>>> child2 = LuckyChild2()
```

```
>>> child2.can_sing()
```

```
Sing a song
```

```
>>> child2.can_dance()
```

```
Shuffle Dance
```

```
>>>
```

물론 굳이 상속이라는 기능을 사용하지 않고도 부모 클래스에 구현된 메서드를 그대로 복사해서 새로 정의할 클래스에 코드를 붙여넣는 식으로 사용할 수도 있습니다. 단, 이렇게 하게 되면 같은 기능을 하는 코드가 중복으로 발생하기 때문에 코드를 관리하기가 어렵고 복사 및 붙여넣기를 해야 하므로 불편합니다. 이에 반해 클래스의 상속이라는 기능을 이용하면 최소한의 코드로도 부모 클래스에 구현된 메서드를 손쉽게 바로 이용할 수 있습니다.

1) 연습문제

문제 6-1

다음의 조건을 만족하는 Point 라는 클래스를 작성하세요.

- Point 클래스는 생성자를 통해 (x, y) 좌표를 입력받는다.
- setx(x), sety(y) 메서드를 통해 x 좌표와 y 좌표를 따로 입력받을 수도 있다.
- get() 메서드를 호출하면 튜플로 구성된 (x, y) 좌표를 반환한다.
- move(dx, dy) 메서드는 현재 좌표를 dx, dy 만큼 이동시킨다.
- 모든 메서드는 인스턴스 메서드다.

문제 6-2

문제 6-1 에서 생성한 Point 클래스에 대한 인스턴스를 생성한 후 4 개의 메서드를 사용하는 코드를 작성하세요.

문제 6-3

아래의 Stock 클래스에 대해 2 개의 인스턴스를 생성했을 때 클래스와 a 와 b 인스턴스의 네임스페이스를 그려보세요.

```
>>> class Stock:
```

```
    market = "kospì"
```

```
>>> a = Stock()
```

```
>>> b = Stock()
```

문제 6-4

문제 6-3 의 코드에서 추가로 아래와 같은 코드를 수행했을 때 '???'로 표시된 부분의 결괏값을 적어보세요.

```
>>> a.market
```

```
???
```

```
>>> b.market
```

```
???
```

```
>>> Stock.market
```

```
???
```

```
>>> a.market = "kosdaq"
```

```
>>> b.market = "nasdaq"
```

```
>>> a.market
```

```
???
```

```
>>> b.market
```

```
???
```

```
>>> Stock.market
```

```
???
```

```
>>>
```

문제 6-5

문제 6-3, 문제 6-4의 코드가 모두 수행된 후의 Stock 클래스, a 인스턴스와 b 인스턴스의 네임스페이스를 그려보세요.