

파이썬 프로그래밍

11주차 강의자료

나재호

CHAPTER
11

클래스와 객체

- 01 객체지향 언어의 개념
- 02 클래스 정의
- 03 객체의 생성
- 04 생성자
- 05 정보은닉: 비공개 속성과 액세스 메서드
- 06 클래스 단위 멤버

이 단원을 마치고 나면

- 클래스와 객체에 대해 그 개념과 역할을 설명할 수 있다.
- 클래스를 정의하여 객체를 생성할 수 있다.
- 생성된 객체의 속성과 메서드에 접근할 수 있다.
- 클래스에 생성자를 추가하고 객체 생성 시 이용할 수 있다.
- 비공개 속성을 지정하고, 이에 접근하는 액세스 메서드를 정의할 수 있다.
- 정적 메서드의 특징을 이해하고 정의할 수 있다.



01 | 객체지향 언어의 개념

프로그래밍 기법

- ◆ 프로그램 = 데이터 + 알고리즘
 - 데이터: 프로그램에서 사용하는 정보
 - 알고리즘: 문제를 해결하는 절차들

- ◆ 프로그래밍 기법: 문제 해결 방식에 따른 분류
 - 절차지향형 프로그래밍 (대표적인 예: C언어)
 - 객체지향형 프로그래밍 (대표적인 예: JAVA 언어)

절차지향 언어(procedure-oriented language)

- ◆ 문제 해결을 위해 필요한 절차들을 나열해 가며 전체 문제를 해결하는 방식의 프로그래밍 언어
 1. 데이터를 먼저 기술
 2. 데이터를 사용하고 조작하는 각종 함수를 정의
 3. 각 과정에 필요한 함수를 순차적으로 호출
 - 각 함수는 앞서 정의한 데이터를 조작해 가며 문제 해결
 - 예: 원의 넓이를 구하는 프로그램

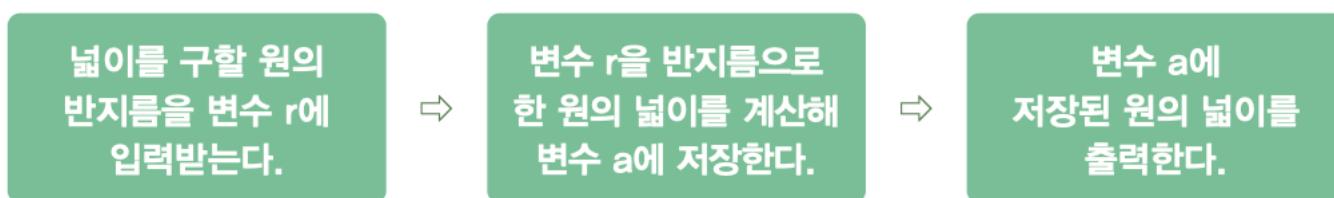


그림 11.1 절차지향적으로 원의 넓이 계산

객체지향 언어(object-oriented language)

- ◆ 객체 간의 상호작용을 통해 전체 문제를 해결하는 방식의 프로그래밍 언어
 - 객체(object): 어떠한 대상에 연관된 정보를 담고 있는 데이터와 이 데이터를 조작하는 함수들을 하나로 묶음
 - 복잡한 프로그램을 명료하고 편리하게 만들 수 있음
 - 예: 원의 넓이를 구하는 프로그램



그림 11.2 객체지향적으로 원의 넓이 계산

절차지향적으로 밥 짓기

- ① 솥을 준비한다.
- ② 솥에 물과 쌀을 적정 비율로 넣는다.
- ③ 솥뚜껑을 닫는다.
- ④ 센 불로 솥을 가열한다.
- ⑤ 한 번 끓어오르면 불의 세기를 중간 불로 줄인다.
- ⑥ 약 10~15분 후 불을 완전히 끈다.
- ⑦ 약 10분 정도 뜸을 들인다.
- ⑧ 솥뚜껑을 열고 밥을 얻는다.

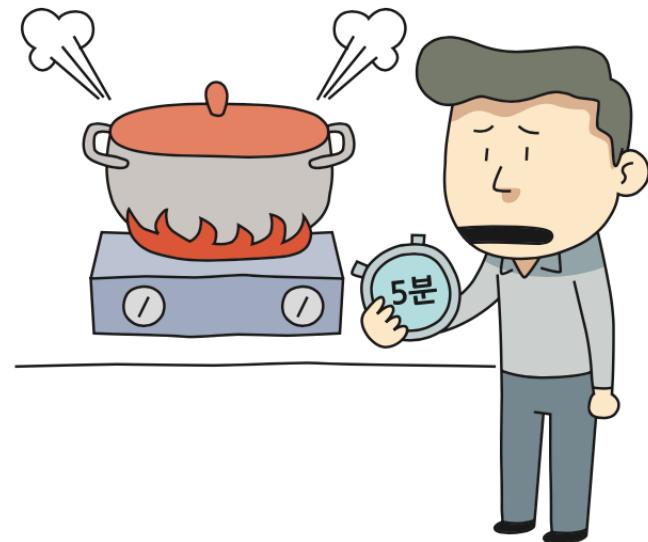


그림 11.3 일반 솥으로 밥 짓기

- 전 과정을 매번 세심히 살펴야 하는 번거로움
- 때에 따라서 죽밥, 설익은 밥, 혹은 누룽지가 될 수도

객체지향적으로 밥 짓기

- ① 밥을 지을 수 있는 전기밥솥을 준비한다.
- ② 전기밥솥에 물과 쌀을 적정 비율로 넣는다.
- ③ 전기밥솥의 뚜껑을 닫는다.
- ④ 전기밥솥에 전원을 연결하고 취사 버튼을 누른다.
- ⑤ 전기밥솥의 상태가 보온 상태가 될 때까지 기다린다.
- ⑥ 전기밥솥 뚜껑을 열고 밥을 얻는다.



그림 11.4 전기밥솥으로 밥 짓기

- 밥 짓는 과정에 관여할 필요 없음
- 항상 일정한 수준의 밥을 얻을 수 있음

객체는 어디서 구하나?

◆ 객체지향적으로 밥을 짓고자 하는데 전기밥솥이 없다면?

- 반드시 먼저 내 문제를 해결할 때 필요한 객체(전기밥솥) 준비 필요
 - 구매하거나
 - 혹은 직접 제작해야 함



그림 11.5 객체를 준비하자

객체지향 언어는 복잡만 할까?

- ◆ 상대적으로 초기에는 복잡해 보이고 비효율적인 것처럼 느껴질 수 있음
 - 당장 밥이 필요한데 전기밥솥을 만들어야 함
 - 더불어 전기밥솥을 만들려면 밥 짓는 과정도 다 알아야 함
 - 전기밥솥을 만드는 대신 그냥 밥을 지으면 안되나?
- ◆ 그러나 사용 가능한 객체들이 늘어난다면?
 - 손쉽게 요리 가능



그림 11.6 어떻게 요리할 것인가?

객체지향적으로 프로그래밍 하자

◆ 객체지향 언어

- 연관된 데이터와 함수들을 객체라는 모듈화된 한 덩어로 묶어 줌
- 내 문제를 해결하기 위해 필요한 코드를 더 명확하고 쉽게 작성할 수 있게 기여함

◆ 초기에는 상대적으로 더 느린 처리 속도로 외면

- H/W 기술의 발전으로 오늘날에는 처리 속도의 차이가 무시할 만함
- 객체지향 언어가 갖는 장점이 더 부각되어 대부분의 프로그래밍 언어들이 객체지향적인 특징을 지님

객체(object)란 무엇인가?

- ◆ 현실세계의 사물 또는 대상(object)을 모방한 프로그램상의 대상(물)을 가리킴

object



1. 명사 물건, 물체

2. 명사 욕망, 연구, 관심 등의 대상

객체² (客體)



[명사]

1. [철학] 의사나 행위가 미치는 대상.

2. [언어] 문장 내에서 동사의 행위가 미치는 대상.

3. [철학] 작용의 대상이 되는 쪽.

출처: 옥스퍼드 영한사전

출처: 국립국어원 표준국어대사전

- ◆ 객체(Object) = 데이터 + 알고리즘

- 자신이 처리할 '데이터'를 가짐
- 자신이 제공하는 '기능'을 통해 외부세계의 다른 객체와 상호작용
- 추상적으로 표현

객체 지향(object-oriented) 프로그래밍

- ◆ 현실에 존재하는 사물과 대상의 상태, 그리고 그에 따른 행동을 실체화시키는 형태의 프로그래밍
 - 하나의 프로그램을 상호 연결된 객체들의 집합으로 봄
 - 단순 절차(알고리즘) 보다 객체를 중시
 - 문제와 관련된 객체를 만들고 이 객체들 간의 메시지 교환으로 문제 해결

프로그램

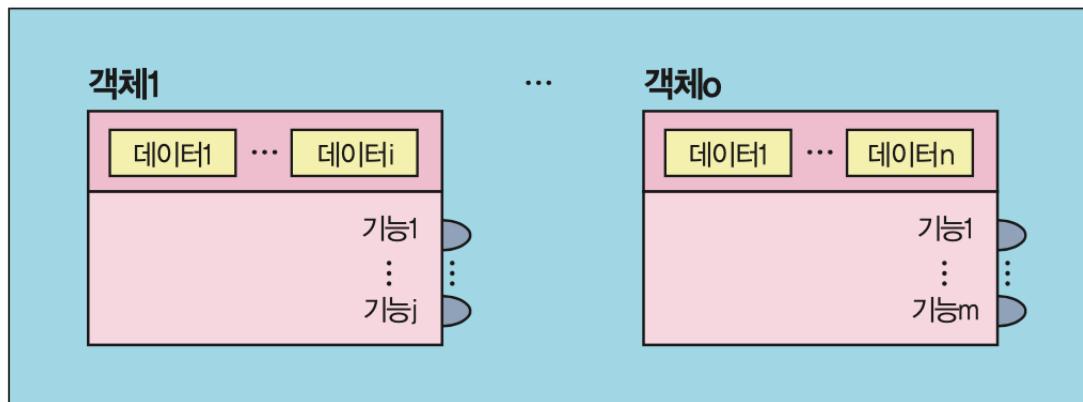


그림 11.7 객체와 프로그램의 구성



02 | 클래스 정의

객체지향적으로 접근하기

- ◆ 다음 상황을 시뮬레이션 하는 프로그램 작성을
객체지향적으로 작성해 보자

문제

“반지름이 1인 원과 반지름 10인 원의 넓이와 둘레를 각각 구하시오.”

- 객체를 이용해 문제를 해결하는 것에 대한 맛보기

만일 절차지향적으로 작성한다면

코드 11-1

절차지향적으로 원의 둘레와 넓이 구하기

```
01 # 사용자 정의 함수부
02 def get_circumference(radius) :
03     result = 2 * 3.14 * radius
04     return result
05
06 def get_circle_area(radius) :
07     result = 3.14 * radius ** 2
08     return result
09
10 # 주 프로그램부
11 small = 1
12 c = get_circumference(small)
13 a = get_circle_area(small)
14 print(f'반지름 {small}인 원의 둘레는 {c:.2f}이고, 넓이는 {a:.2f}이다.')
15
16 big = 10
17 print(f'반지름 {big}인 원의 ', end='')
18 print(f'둘레는 {get_circumference(big):.2f}이고, ', end='')
19 print(f'넓이는 {get_circle_area(big):.2f}이다.')
```

객체의 구성요소

◆ 속성(Attributes)

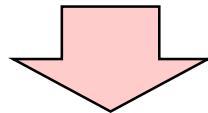
- 객체의 특성을 표현
 - 현재 객체 자신의 상태 정보를 표현
- 정적인 성질

◆ 행위(Behaviors)

- 객체 내부에서 처리할 일이나, 객체들간에 영향을 주고 받는 일
- 동적인 일을 처리하는 단위

문제로부터 객체 찾기

- ◆ "반지름 1인 원과 반지름 10인 원의 넓이와 둘레를 각각 구하시오"



- 반지름, 1, 10, 원, 넓이 구하기, 둘레 구하기
 - 임의의 반지름을 가진 원의 넓이와 둘레를 계산하는 객체 필요
 - 각 반지름을 갖는 원 객체를 이용해 처리

원 객체 표현의 예

◆ 속성

- 반지름
 - 단, 각 객체는 다른 '속성값'을 가질 수 있다

◆ 행위

- 둘레 구하기
- 넓이 구하기

반지름: 1

둘레 구하기
넓이 구하기

small

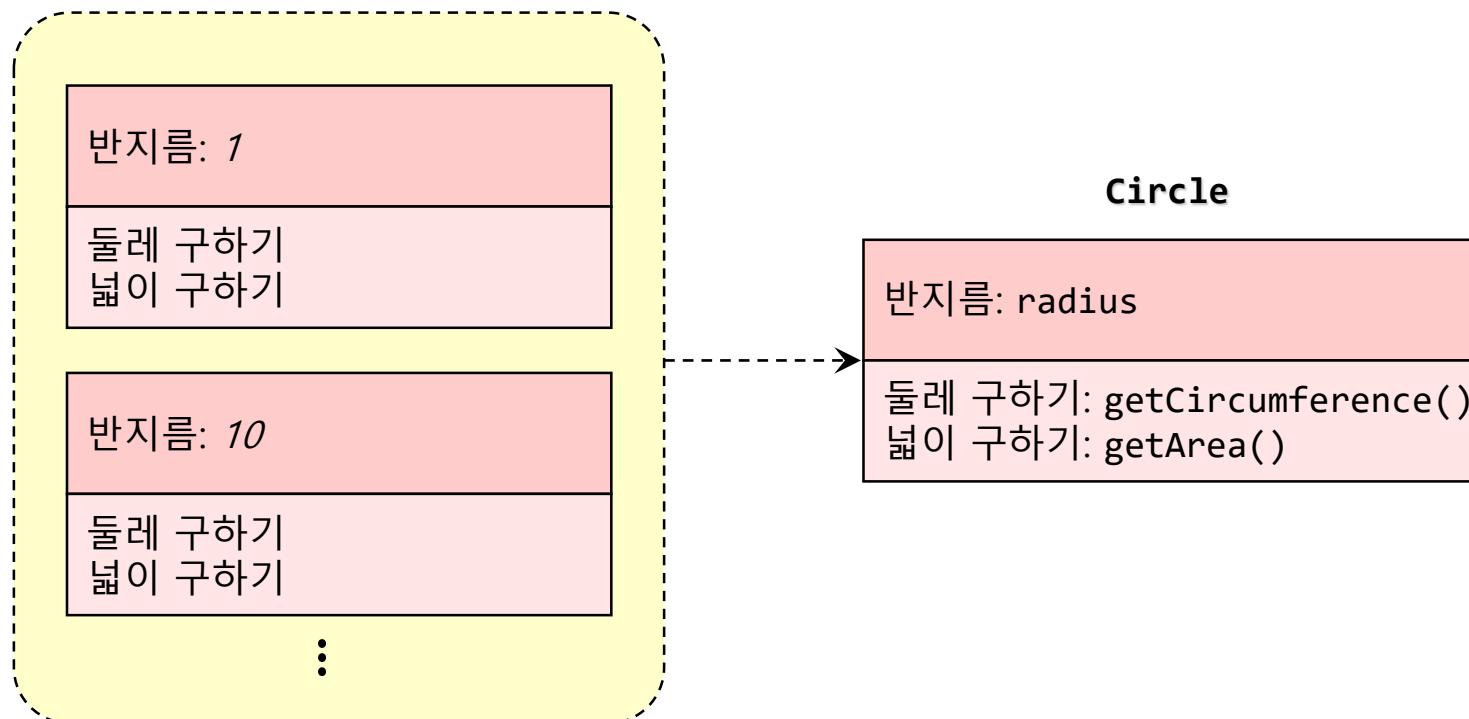
반지름: 10

둘레 구하기
넓이 구하기

big

객체 간의 공통된 구조와 기능

- ◆ 같은 종류의 객체의 공통된 '데이터 구조'와 '기능'을 따로 뽑아 틀(클래스)로 정의



클래스(class)

- ◆ 같은 종류의 객체 생성을 위해 정의되는 틀
 - 그 클래스로부터 생성될 객체들의 공통된 속성과 행위를 추상적으로 표현하여 정의
 - 단, 같은 클래스로 만든 객체일 지라도 각 객체는 서로 다른 속성값을 지닐 수 있음
- 클래스 정의는 단지 틀(껍데기)일뿐!

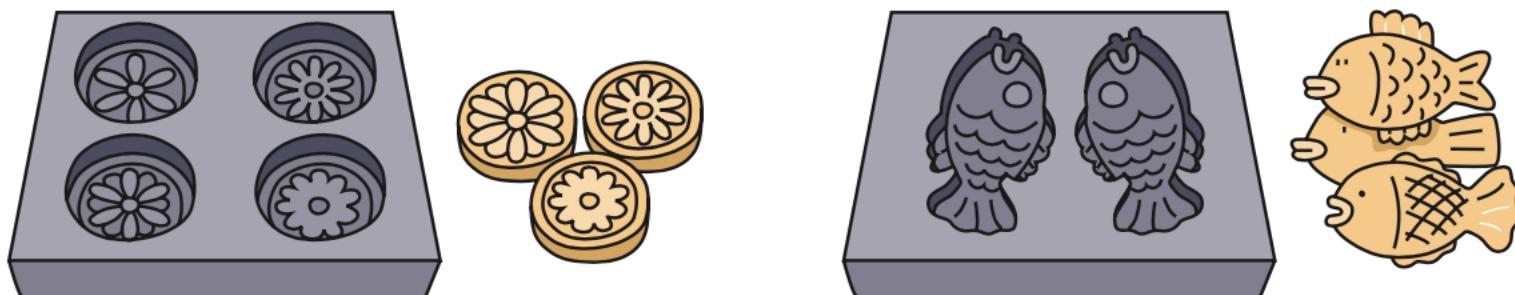


그림 11.9 다양한 빵틀과 빵

클래스 vs. 객체

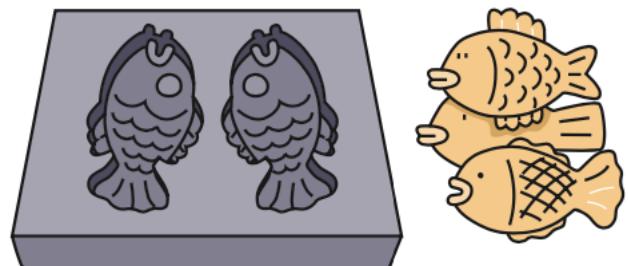
◆ 클래스

- 객체를 만들기 위한 '틀'
- 객체가 지니는 '속성'과 '행위'를 추상적으로 정의

◆ 객체

- 해당 클래스의 구체적인 한 실례(instance)
- 같은 클래스로 만들어진 각 객체는
 - 속성에 고유한 값을 지닐 수 있음
 - 동일한 행동을 할 수 있음

◆ 예: "빵틀"과 "빵"



클래스의 정의

- ◆ 객체 생성 전에 먼저 정의되어야 함
 - 객체가 어떤 속성을 갖고 어떤 행위를 할지 기술

```
class 클래스이름 :  
    데이터_속성_정의  
    메서드_정의
```

- 데이터 속성 정의는 명시적으로 선언되지 않고 생략 가능



두 메서드를 갖는 클래스 정의의 양식

```
class 클래스이름 :  
    def 메서드_이름1(self [, ...]) :  
        처리할_문장1  
  
    def 메서드_이름2(self [, ...]) :  
        처리할_문장2
```

- 이 클래스로부터 만들어진 객체가 지닐 메서드(기능을 기술한 함수)에 대한 정의 포함
- 각 메서드는 첫 번째 매개변수로 `self`를 가짐
 - 관례적인 이름
 - 클래스로부터 만들어진 객체의 인스턴스 자신을 참조
 - 이 변수로 클래스 내에 정의한 다른 멤버에 접근 가능

Circle 클래스의 기본 구조

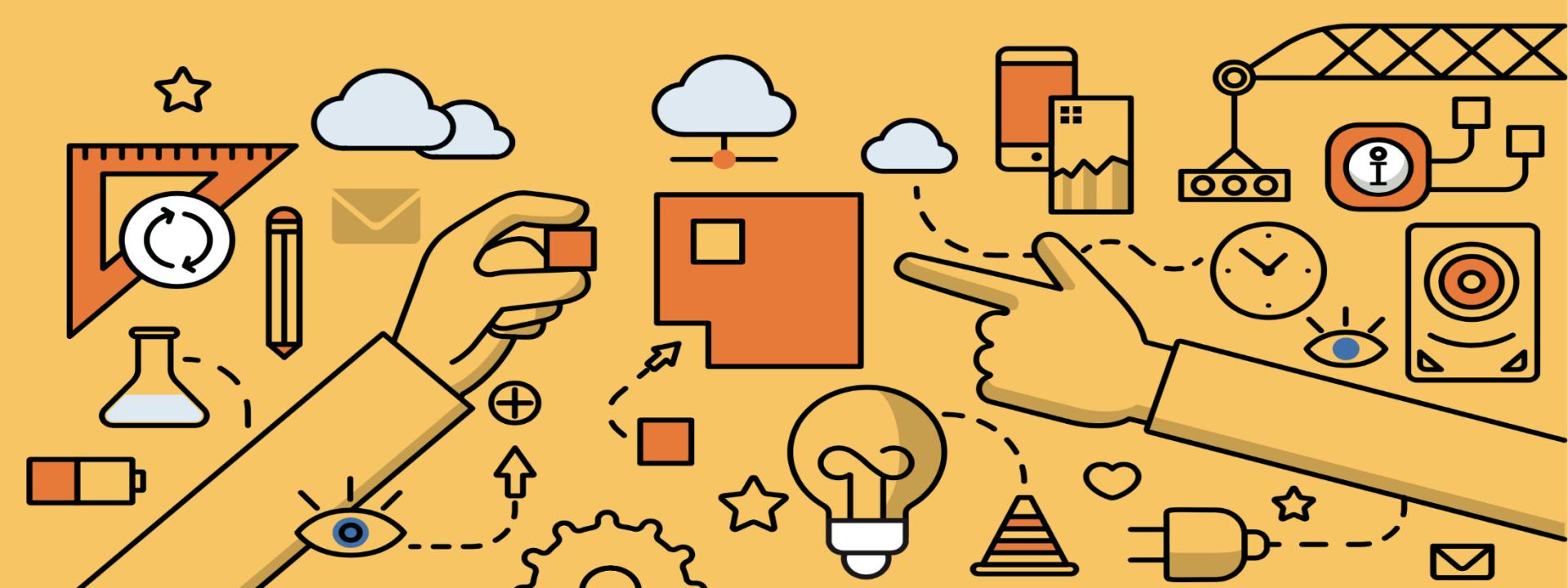
#

코드 11-2

의사 코드: 원 객체 생성을 위한 클래스

```
01 class Circle :  
02     def getCircumference(self) :  
03         pass  
04  
05     def getArea(self) :  
06         pass
```

- 일단 클래스 구조에 초점을 맞추기 위해 각 메서드의 세부 구현은 하지 않음
- 다만 하나의 틀인 클래스 정의만으로는 어떤 처리도 할 수 없다!



03 | 객체의 생성

정의한 클래스로 객체를 만들자

- ◆ 클래스를 정의했다면 구체화된 실체인 객체를 생성해야 문제 처리에 사용 가능
- ◆ 파이썬에서 객체 생성은 함수 호출 표기법 사용

```
객체이름 = 클래스이름()
```

원 객체의 생성

#

코드 11-3

원 객체 생성

```
small = Circle()  
big = Circle()
```

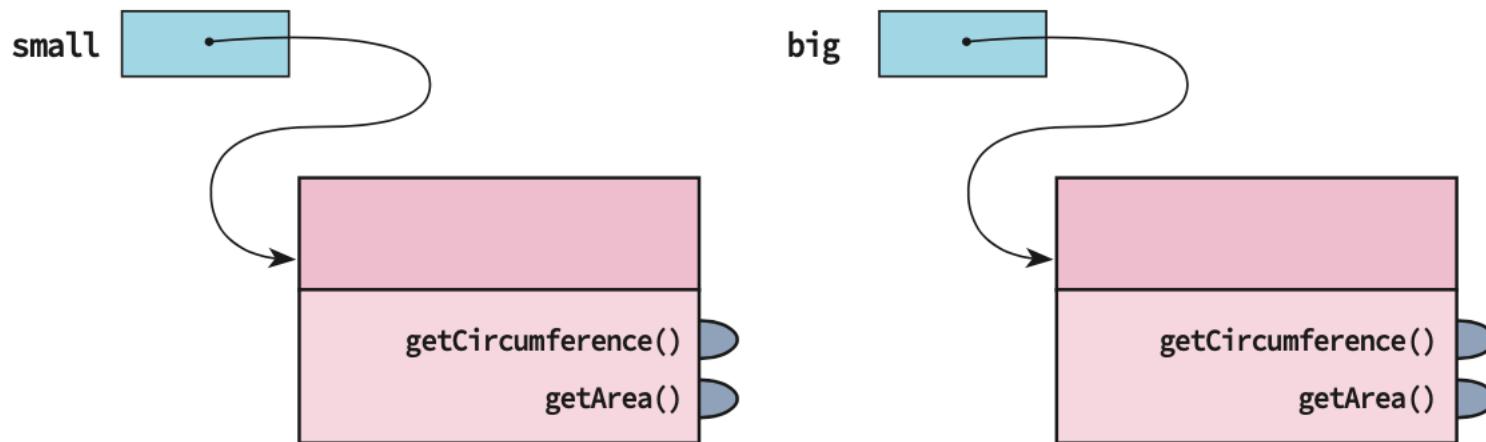


그림 11.11 아직은 미흡한 상태의 원 객체

- 각 객체는 독립적인 메모리 공간에 생성 됨

객체에 속성 추가하기

- ◆ 원의 넓이/둘레를 구하기 위해선 반지름을 알아야 함
 - 일반 함수로 구현 시 인수로 전달
 - 연관된 정보/기능의 묶음인 객체가 반지름을 알고 있다면 메서드 호출 시 전달할 필요 없음

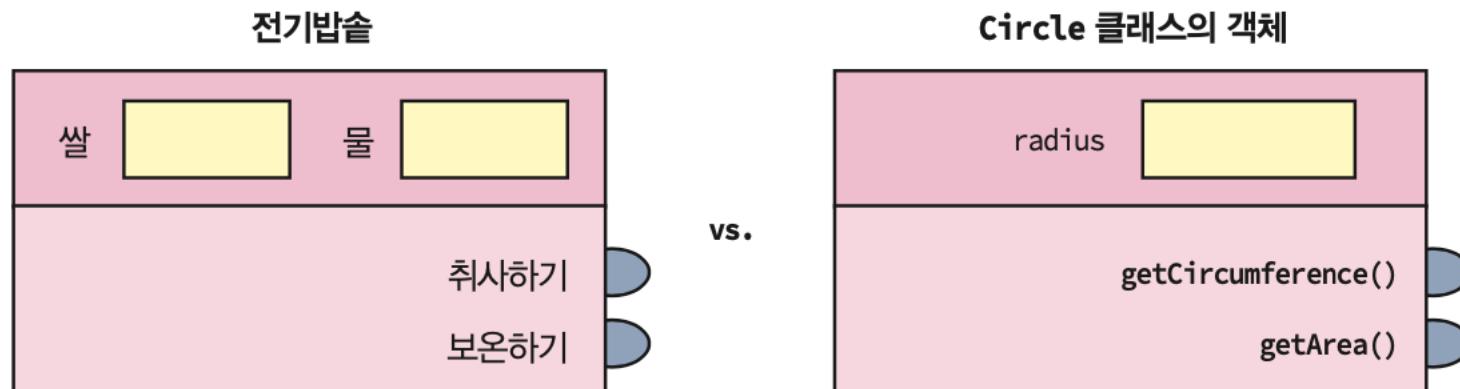


그림 11.12 전기밥솥과 Circle 클래스 객체의 비교

객체의 속성 추가 및 접근

- ◆ 객체에 대한 변수에 속성 참조 연산자 . 을 사용

```
객체이름.데이터속성이름 = 값
```

- 대상 객체에 새로운 데이터 속성 추가 가능
 - 파이썬에서는 값 할당을 통해 변수가 선언되므로
- 대상 객체가 존속하는 한 계속 유지됨
 - 대상 객체의 기존의 데이터 속성 값 수정
 - 이 객체의 다른 메서드에 의해서도 접근 가능

두 원 객체의 생성

#

코드 11-4 객체에 데이터 속성 추가

```
01 class Circle :  
02     def getCircumference(self) :  
03         pass  
04  
05     def getArea(self) :  
06         pass  
07  
08 small = Circle()  
09 big = Circle()  
10  
11 small.radius = 1      # 객체 small의 반지름 속성을 1로 설정  
12 big.radius = 10       # 객체 big의 반지름 속성을 10으로 설정
```

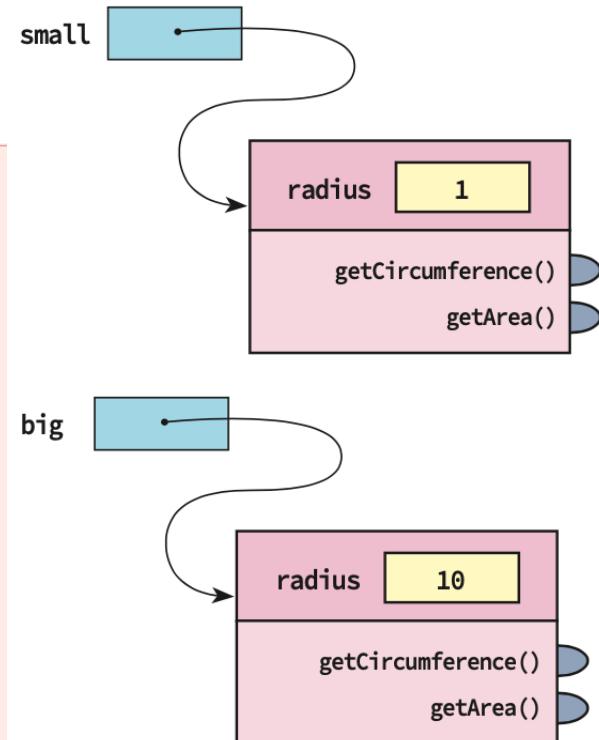


그림 11.13 온전하게 생성된 원 객체

메서드에서 자신의 속성에 접근하기

- ◆ 각 메서드는 호출 시 자동으로 자신을 호출한 객체에 대한 아이디를 self 매개변수로 전달 받음
 - 각 메서드는 객체 간의 공통의 공간에 할당됨
 - self를 이용해 자신의 다른 속성에 접근 가능

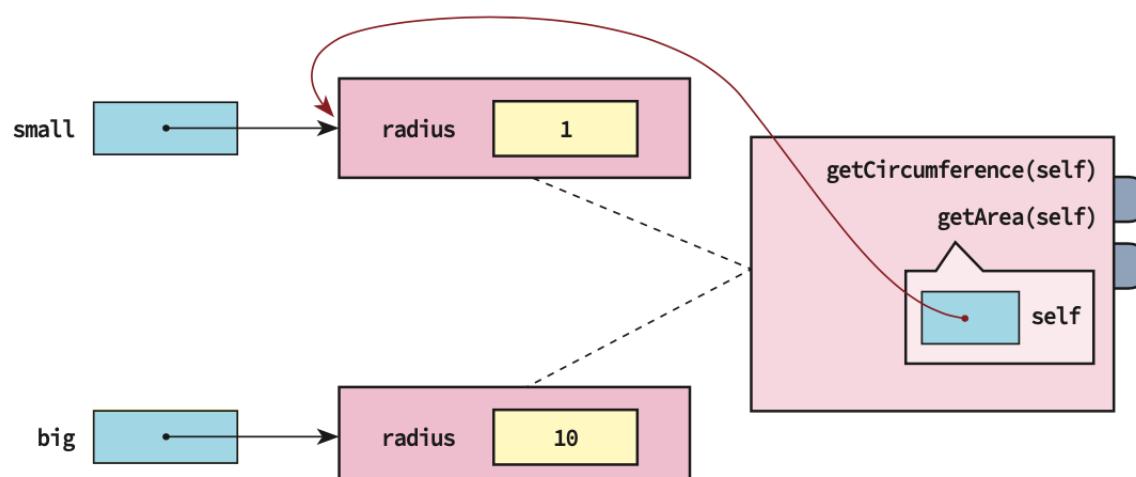


그림 11.14 small.getArea() 호출 시 매개변수 self의 역할

원 클래스 정의의 완성

코드 11-5

Circle 클래스를 이용한 둘레와 넓이 구하기

```
01 # 원 클래스 정의부
02 class Circle :
03     def getCircumference(self) :
04         result = 2 * 3.14159265 * self.radius
05         return result
06
07     def getArea(self) :
08         result = 3.14159265 * self.radius ** 2
09         return result
10
11 # 주프로그램 부
12 small = Circle()
13 big = Circle()
14
15 small.radius = 1      # 객체 small의 반지름 속성을 1로 설정
16 big.radius = 10       # 객체 big의 반지름 속성을 10으로 설정
```

자신의 radius라는
데이터 멤버의 값을 참조하라

원 객체의 생성과 사용

- ◆ 메서드 호출 시 self에 대한 인수는 내부적으로 자동 전달됨

#

코드 11-5

Circle 클래스를 이용한 둘레와 넓이 구하기 (cont)

```
18 print(f'반지름 {small.radius}인 원의 ', end=' ')
19 print(f'둘레는 {small.getCircumference():.2f}이고, ', end=' ')
20 print(f'넓이는 {small.getArea():.2f}이다.')
21
22 print(f'반지름 {big.radius}인 원의 ', end=' ')
23 print(f'둘레는 {big.getCircumference():.2f}이고, ', end=' ')
24 print(f'넓이는 {big.getArea():.2f}이다.'
```



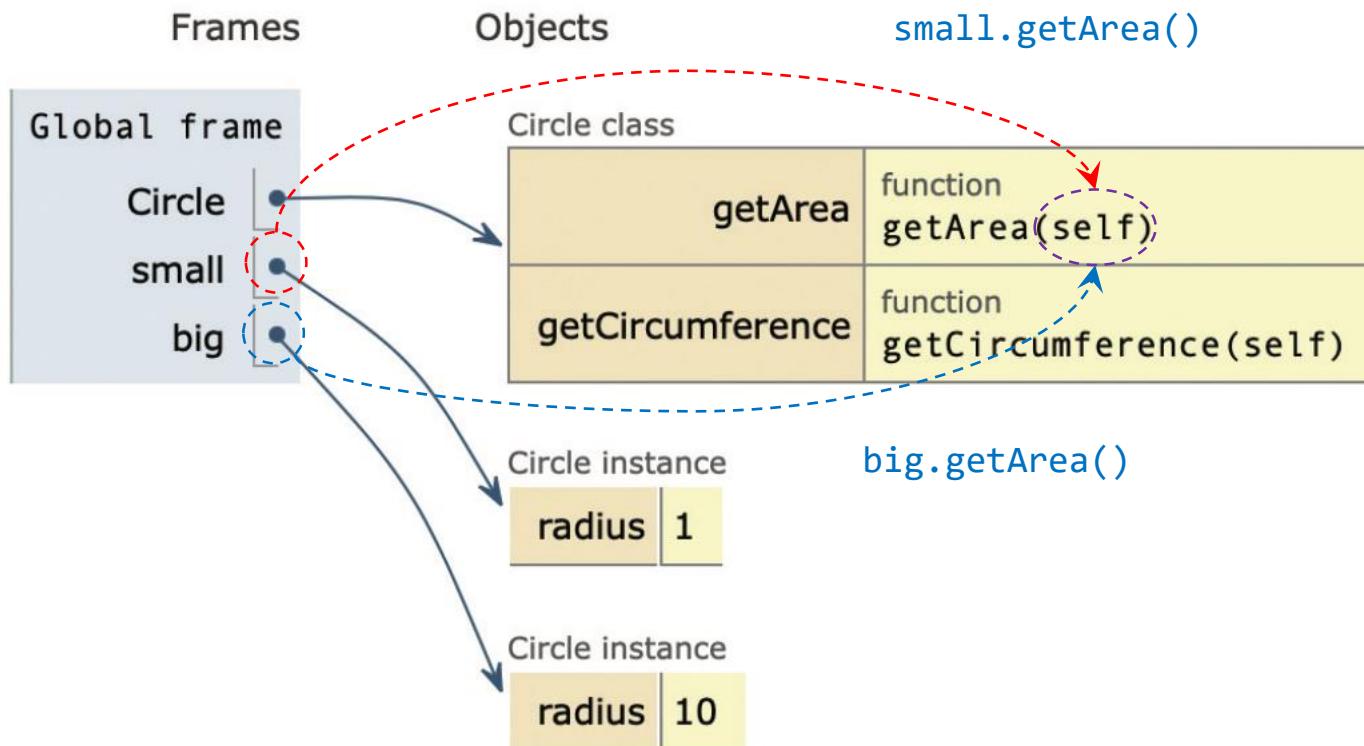
실행 결과

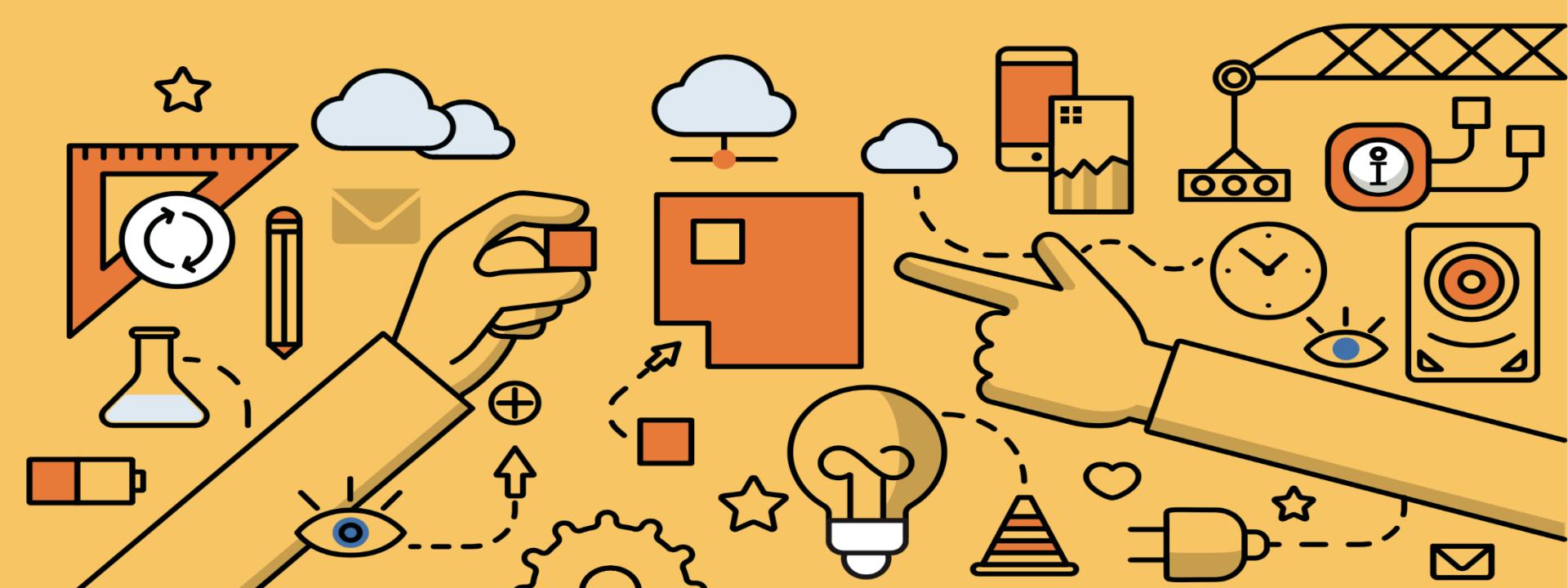
반지름 1인 원의 둘레는 6.280이고, 넓이는 3.140이다.

반지름 10인 원의 둘레는 62.800이고, 넓이는 314.160이다.

파이썬 컴퓨터로 살펴 본 객체의 구성

- ◆ 메서드 호출 시 각 객체의 아이디가 self로 전달
 - 하나의 메서드가 서로 다른 객체의 속성값 사용 가능





04 | 생성자

객체 초기화의 다른 방법

- ◆ 각 객체는 자신에게 부여된 책임에 따른 동작을 **스스로 행할 수** 있도록 작성하는 것이 바람직
 - 안전하고 편리한 사용을 위해
 - 전기밥솥을 이용할 때 열/압력의 세기를 사람이 직접 조정하지 않는다!
- ◆ 객체 사용을 위한 초기화 역시 **객체 스스로** 하는 것이 바람직
 - 세탁기의 경우 물/세제의 양이 자동으로 주입되듯이
 - 그런데, 코드 11-5의 15~16행은 객체 외부에서 객체를 초기화 하고 있음

```
15 small.radius = 1  
16 big.radius = 10
```

생성자(Constructor)

- ◆ 객체가 생성될 때 단 한 번 자동으로 호출되는 메서드
 - 객체의 속성을 '초기화'하는 역할 수행
 - 명시적으로 따로 호출하지는 않음

- ◆ 생성자 구현 조건
 - 이름이 `_init_()` 인 메서드
 - 반환값을 갖지 않음
 - 메서드이므로 `self` 매개변수를 가짐
 - (필요시) 설정값을 전달받기 위해 추가 매개변수를 가질 수 있음

생성자를 갖는 클래스의 객체 생성

#

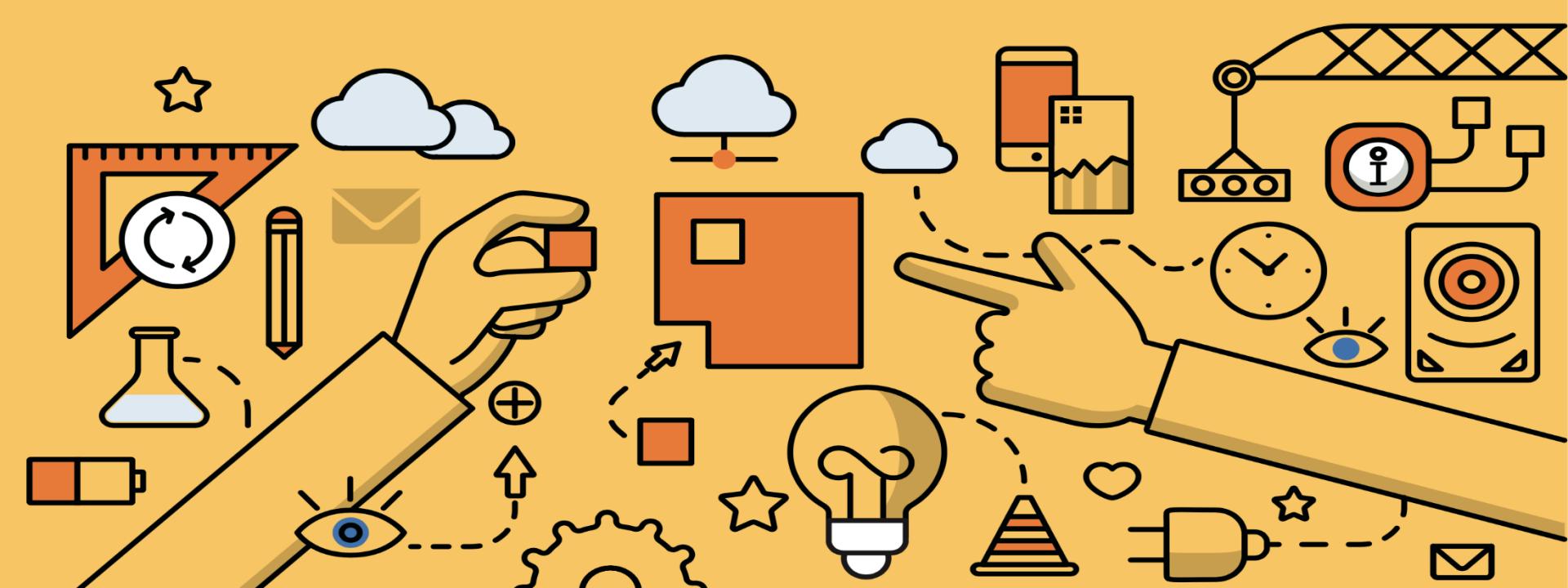
코드 11-6

생성자를 갖는 원 클래스의 정의와 사용

```
01 # 원 클래스의 정의
02 class Circle :
03     def __init__(self, radius) :
04         self.radius = radius
05
06     def getcircumference(self) :
07         result = 2 * 3.14159265 * self.radius
08         return result
09
10    def getArea(self) :
11        result = 3.14159265 * self.radius ** 2
12        return result
13
14 # 주 프로그램부
15 small = Circle(1)      # 반지름 1인 원 객체
16 big = Circle(10)       # 반지름 10인 원 객체
```

생성자 사용의 이점

- ◆ 객체를 생성하면서 동시에 초기화할 수 있음
 - 객체의 속성에 대한 외부로부터의 직접 접근을 피할 수 있음
 - 초기화 과정 누락으로 초기화되지 않은 객체가 존재하게 되는 문제를 방지
 - 객체 사용 중 불필요한 시점에 객체가 초기화되는 문제를 피할 수 있음
 - 단순 할당문이 아니라 메서드를 통해 초기화 작업을 기술할 수 있음
 - 필요시 초기값의 유효성을 확인하여 안전한 초기화가 가능



05

정보 은닉: 비공개 속성과 액세스 메서드

배경

- ◆ 기본적으로 객체의 속성은 객체 외부에서의 접근이 언제든지 허용됨
 - 생성자를 통해 객체 속성의 직접 생성을 막더라도, 만들어진 속성에 대한 접근을 막지는 않음

```
22 print(f'반지름 {big.radius}인 원의 ', end='')
```

```
23 print(f'둘레는 {big.getCircumference():.2f}이고, ', end='')
```

```
big.radius = -10
```

```
24 print(f'넓이는 {big.getArea():.2f}이다. ')
```

- ◆ 정보은닉의 필요성

- 객체의 속성이나 세부 구현은 굳이 객체 외부에 노출하지 않도록 하자

비공개 속성

- ◆ 해당 클래스 객체의 메서드에서만 접근을 허용하고, 객체 외부에서는 직접 접근할 수 없도록 **숨김**
 - 정보 은닉의 관점
 - '__속성이름'과 같이 식별자 부여
 - 식별자 앞쪽에만 __를 붙인 경우에 한정됨
 - 객체 내부적으로 '__클래스이름__속성이름__'의 형태로 자동 변환되어
 - 외부에서 '__속성이름'으로 접근을 차단

비공개 속성을 갖는 클래스의 예

코드 11-7

비공개 속성을 갖는 클래스

```
01 # 원 클래스의 정의
02 class Circle :
03     def __init__(self, radius) :
04         self.__radius = radius
05
06     def getcircumference(self) :
07         result = 2 * 3.14159265 * self.__radius
08         return result
09
10    def getArea(self) :
11        result = 3.14159265 * self.__radius ** 2
12        return result
13
14 # 주 프로그램
15 big = Circle(10)
16 big.__radius = -10      # 비공개 속성에 접근하는 표현이 아님
17 print(f'반지름 {big.__radius}인 원의 ', end='')
18 print(f'둘레는 {big.Circumference():.2f}이다.')
```

액세스 메서드

- ◆ (비공개) 속성을 객체 외부에서 접근할 필요가 있을 때 제공
 - 게터(getter)
 - 설정된 값을 얻어와 주는 메서드
 - 해당 속성의 속성값을 반환
 - get속성이름()
 - 세터(setter)
 - 지정된 속성에 새로운 값을 설정하는 메서드
 - 보통 해당 속성의 설정값을 인수로 가짐
 - set속성이름()

액세스 메서드의 사용 예

코드 11-8

비공개 속성을 갖는 클래스

```
01 # 원 클래스의 정의
02 class Circle :
03     def __init__(self, radius) :
04         self.__radius = radius
05
06     def getcircumference(self) :
07         result = 2 * 3.14159265 * self.__radius
08         return result
09
10    def getArea(self) :
11        result = 3.14159265 * self.__radius ** 2
12        return result
13
14    def setRadius(self, radius) :
15        self.__radius = radius
16
17    def getRadius(self) :
18        return self.__radius
19
20 # 주 프로그램부
21 big = Circle(10)
22 big.setRadius(100)
23 print(big.getRadius())
```

```
14 def setRadius(self, radius) :
15     self.__radius = radius
16
17 def getRadius(self) :
18     return self.__radius
```



실행 결과

100

보다 나은 세터 구현 사례

- ◆ 무조건 값을 설정하기 보다, 설정 값의 유효성 여부를 판단하여 선택 처리
 - 반환값의 예외처리를 통해 오류 상황 분별 가능

#

코드 11-9

참고: 세터의 구현 예

```
14     def setRadius(self, radius) :  
15         if radius <= 0 :  
16             return False  
17  
18         self.__radius = radius  
19         return True
```



06

클래스 단위 멤버

클래스 단위 멤버

◆ 데이터 속성

- 인스턴스 변수
 - 각 객체의 고유한 상태값 저장, 각 객체마다 개별 공간 할당
 - self 매개변수로 참조됨
- 클래스 변수

◆ 메서드

- (인스턴스) 메서드
 - self 매개변수를 가짐
- 정적 메서드
- 클래스 메서드

클래스 변수

◆ 클래스 정의 내에 직접 기술된 변수

- 이 클래스의 클래스 간의 공통 영역에 할당됨
 - 이 클래스의 모든 객체에서 **공유**됨
- 객체 생성과 무관하게 사용 가능
 - 클래스 이름을 통해 접근

클래스이름.클래스변수이름

비효율적으로 공통의 정보를 갖는 객체들

코드 11-10 공통의 변수를 갖는 객체들

```
01 # 원 클래스의 정의
02 class Circle :
03     def __init__(self, radius) :
04         self.__PI = 3.14159265
05         self.__radius = radius
06
07     def getCircumference(self) :
08         result = 2 * self.__PI * self.__radius
09         return result
10
11     def getArea(self) :
12         result = self.__PI * self.__radius ** 2
13         return result
14
15     ...
16
17     # 주 프로그램부
18     small = Circle(1)      # 반지름 1인 원 객체
19     big = Circle(10)       # 반지름 10인 원 객체
```

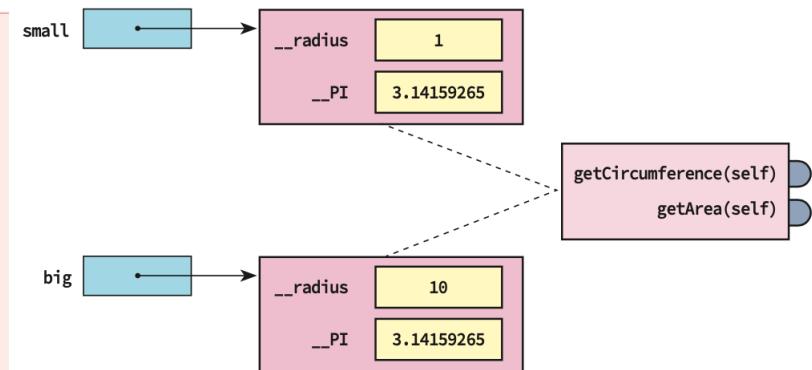


그림 11.16 공통의 변수를 갖는 객체들

클래스 변수를 갖는 클래스 정의의 예

#

코드 11-11 클래스 변수를 갖는 클래스 정의

```
01 # 원 클래스의 정의
02 class Circle :
03     __PI = 3.14159265
04
05     def __init__(self, radius) :
06         self.__radius = radius
07
08     def getcircumference(self) :
09         result = 2 * Circle.__PI * self.__radius
10         return result
11
12     def getArea(self) :
13         result = Circle.__PI * self.__radius ** 2
14         return result
15
16     :
17
22 # 주 프로그램부
23 small = Circle(1)      # 반지름 1인 원 객체
24 big = Circle(10)       # 반지름 10인 원 객체
```

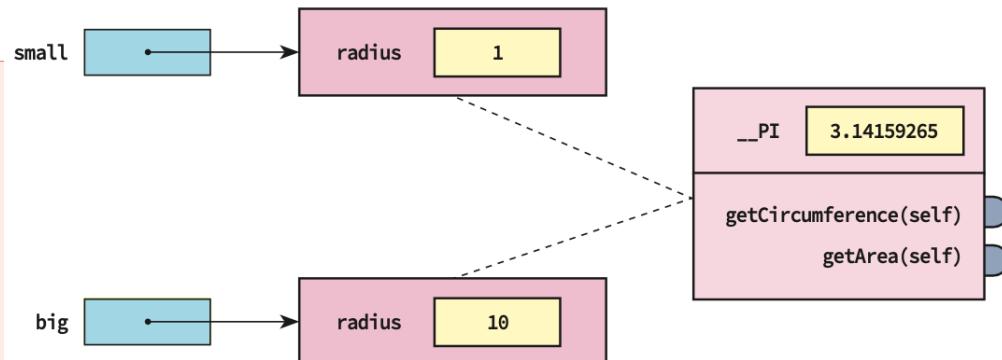


그림 11.17 클래스 변수의 사용



궁금합니다

```
import math
print(math.pi)
```

정적 메서드와 클래스 메서드

◆ 특정 객체와 무관하게 접근해 사용할 수 있는 메서드

- 객체 생성 전에도 호출 가능
 - 매개변수 `self`를 갖지 않음
- 메서드 선언 앞에 데코레이터 추가하여 선언

정리

정적 메서드: 메서드 선언 앞에 `@staticmethod` 데코레이터 추가

클래스 메서드: 메서드 선언 앞에 `@classmethod` 데코레이터 추가

정적 메서드

- ◆ 객체 생성 없이도 클래스 이름을 통해 호출 가능

#

코드 11-13

정적 메서드와 클래스 메서드

```
01 # 원 클래스의 정의
02 class Circle :
03     __PI = 3.14159265
04
05     @staticmethod
06     def getPI() :
07         return Circle.__PI
08
09     # ... 생략 ...
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 # 주 프로그램부
27 print(Circle.getPI())
```



실행 결과

3.14159265

연습문제 11.1

- ◆ 연관된 정보로 2차원 상의 한 좌표점(x, y)를 객체로 표현
 - Point 객체 생성 시 생성자의 인수로 두 정수를 지정
순서대로 이를 (x, y)값으로 설정. 인수값 생략시 기본값 (0, 0)
 - 메소드 show(): 현재 객체의 좌표값을 (x, y) 형식의 문자열로 출력
 - 메소드 set(): Point 객체에 새로운 좌표값 설정
 - 메소드 get(): 현재 객체가 갖고 있는 좌표값을 튜플로 반환

연습문제 11.1

- ◆ 연관된 정보로 2차원 상의 한 좌표점(x, y)를 객체로 표현

- 아래 test코드 사용

```
#사용자 정의 함수부
```

```
def test():
    p1 = Point()
    p2 = Point(2, 3)
```

```
p1.show();
```

```
p1.set(10, 20); p1.show();
```

```
p2.show();
```

```
x, y = p2.get()
print(f'x={x}, y={y}')
```

```
# 주 프로그램부
```

```
if __name__ == '__main__':
    test()
```

- 실행 결과

(0, 0)

(10, 20)

(2, 3)

x=2, y=3

연습문제 11.2

◆ 시각을 나타내는 Time 클래스 정의 및 테스트

- 객체 생성시 설정할 시와 분을 인수로 전달받는 생성자 정의
 - 시와 분에 대한 인수가 생략되면 기본값 0으로 지정
 - 유효한 범위의 정수만 전달된다 가정
 - 작성 예시) `t1 = Time(9); t2 = Time(9, 30)`
- 메소드 `display()`: Time 객체의 시각 출력
 - 작성 예시) `t1.display() # 9:00이라 출력`
- 메소드 `add()`: 입력받은 시각을 현재 시각에 더한 결과를 반환
 - 더한 분이 60분을 넘기면, 결과에서 1시간을 더하고 60분을 뺌
예) 1시 50분 + 15분 = 2시 5분
 - 두 시각을 더한 결과는 24시를 넘지 않는다고 가정
 - 작성 예시): `later = t1.add(Time(1,15))`
- 정적 메소드 `isValid()`: 인수로 지정한 시각이 유효한지 판단
 - 시간은 0~23시, 분은 0~59분 사이여야 함
 - 이를 만족하면 `True`를, 만족하지 않으면 `False` 반환

연습문제 11.2

◆ 시각을 나타내는 Time 클래스 정의 및 테스트

- 아래 test코드 사용

#사용자 정의 함수부

```
def main():
    t1 = Time(9)
    t2 = Time(9, 30)

    t1.display()
    t2.display()

    later = t1.add(Time(1, 15))
    later.display()
```

```
if Time.is_valid(25, 0):
    print('유효한 시각')
else:
    print('유효하지 않은 시각')
```

주 프로그램부

```
if __name__ == '__main__':
    main()
```

- 실행 결과

09:00

09:30

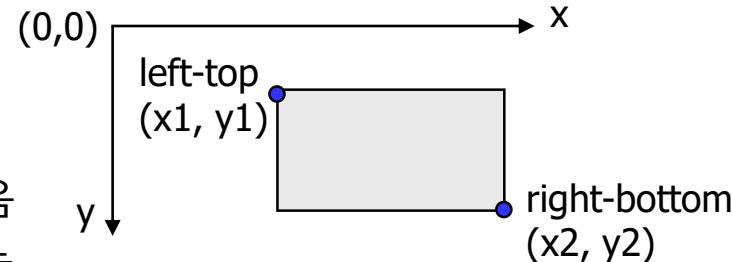
10:15

유효하지 않은 시각

개념 확인 과제(연습문제 11.3)

◆ 2차원 화면에 있는 직사각형의 둘레와 넓이 구하기

- 두 꼭짓점을 연습문제 11.1의 Point 객체로 기술
- Rectangle 클래스 정의
 - 객체 생성시
1, 2번째 인수는 좌측 상단 꼭짓점을,
3, 4번째 인수는 우측 하단 꼭짓점을 받음
 - 인수로 전달받은 값은 클래스 내부에서는
Point 객체 lt와 rb를 통해 유지
 - 구현을 간단히 하기 위해, left-top은 항상 right-bottom보다 왼쪽 상단에 있다
가정 ($x_1 < x_2, y_1 < y_2$)



▪ 위 클래스에 다음 메서드 추가

- show() - 현재 사각형의 좌측 상단과 우측 하단 꼭짓점을 화면에 출력
- getWidth() - 너비 반환, getHeight() - 높이 반환
- getArea() - 넓이 반환, getPerimeter() - 둘레 반환

개념 확인 과제(연습문제 11.3)

◆ 2차원 화면에 있는 직사각형의 둘레와 넓이 구하기

- 아래 테스트 코드 사용

```
# 주 프로그램부
r1 = Rectangle(5, 5, 20, 10)
a = r1.getArea()
p = r1.getPerimeter()

r1.show()
print(f'\n넓이는 {a}, 둘레는 {p}')
```

- 실행 결과

좌측 상단 꼭지점이 **(5, 5)**이고 우측 하단 꼭지점이 **(20, 10)**인 사각형입니다.
넓이는 **75**, 둘레는 **40**

- 제출 기한 및 방법

- 다음 수업시간 전날까지 본인의 repository에 hw9.py를 업로드