<[파이썬 수학] 제16장. 설계도에 생명을! 메소드와 생성자 마법!>

선생님: (지난 시간에 만든 'Car' 설계도와 자동차 인형을 보여주며) 얘들아, 안녕! 지난 시간에 우리는 `Car`라는 클래스(설계도)를 만들고, `my_car = Car()` 처럼 실제 자동차(인스턴스)를 만들었지? 그리고 `my_car.color = "빨강"` 처럼 자동차의 특징(속성)도 정해줬어.

코딩새싹 🌱: 네! 그때는 자동차의 특징만 정했는데, 자동차가 '달린다'거나 '빵빵거린다' 같은 행동은 어떻게 만들어요?

선생님: 아주 좋은 질문이야! 그 '행동'이 바로 클래스 안에 정의하는 함수, 즉 **메소드(Method)**란다! 메소드는 그 설계도로 만들어진 객체(인스턴스)만이 할 수 있는 특별한 능력이 되는 거지.

<1. 인스턴스야, 네 능력을 보여줘! - 메소드 정의하기>

선생님: 메소드를 만들 때는 `def` 키워드를 쓰고, 가장 중요한 첫 번째 재료(파라미터)로 **`self`**를 꼭 넣어줘야 해. 이 `self`가 바로 "이 능력을 사용하는 인스턴스 자기 자신"을 의미한단다!

```
class Book:
  author = "" # 클래스 속성들 (기본값)
  title = ""
  page = 0

  # 책 정보를 출력하는 메소드
  # self는 이 메소드를 호출한 book1 인스턴스 자신이 돼!
  # local은 이 메소드가 추가로 받는 재료(파라미터)야.
  def print_book_info(self, local):
    print(f"저자: {self.author}, 제목: {self.title}, 페이지: {self.page}, 지역: {local}")

  # 간단한 인사말을 출력하는 메소드
  def print_hello(self):
    print("Hello World, 이 책 정말 재미있어요!")

# Book 설계도로 book1 이라는 실제 책을 만들자!
book1 = Book()
```

```
book1.author = "왕새우" # book1의 속성들을 정해주고
book1.title = "파이썬 대모험"
book1.page = 300

# book1아, 너의 정보 출력 능력을 보여줘! (local 재료로 "korea"를 전달)
book1.print_book_info("korea")
# book1아, 인사 한번 해봐!
book1.print_hello()
```

궁금이 💡: `self.author` 처럼 `self` 다음에 점을 찍고 속성 이름을 쓰면, 그 메소드를 부른 인스턴스의 속성값을 가져오는 거군요! 만약 book2가 `print_book_info`를 불렀다면 `self`는 book2가 되는 거고요!

선생님: 정확해! `self` 덕분에 메소드는 "누구의" 정보를 가지고 일해야 하는지 알 수 있는 거란다.

〈2. 객체의 탄생 의식! `__init__` 생성자 함수〉

선생님: 우리가 책(인스턴스)을 만들 때마다 `book1.author = "왕새우"` 이렇게 하나하나 속성을 정해주는 게 조금 번거롭지 않니? 마치 아기가 태어날 때마다 이름표를 따로 붙여주는 것처럼 말이야. "이 아기는 태어나자마자 이름은 '홍길동', 몸무게는 3kg으로 정한다!" 이렇게 **태어나는 순간(인스턴스가 생성되는 순간)에 필요한 설정을 자동으로 해주는** 특별한 의식(메소드)이 바로 `__init__` 이란다!

`__init__`은 '초기화(initialize)'라는 뜻이고, 앞뒤에 밑줄 두 개씩(`__`)이 붙은 아주 특별한 메소드야. 이걸 **생성자(Constructor)**라고도 불러.

```
class Book:
  # 책이 만들어질 때(인스턴스 생성 시) 자동으로 호출되는 __init__ 메소드!
  def __init__(self, author, title, pages):
    # self는 새로 만들어지는 책(인스턴스) 자기 자신!
    # "이 책의 작가는 재료로 받은 author 값으로 한다!"
    self.author = author
    self.title = title
    self.pages = pages
    print(f"'{self.title}' 책 한 권이 뚝딱! 생성되었습니다.") # 생성 메시지!
```

```
  def print_book_info(self):
    print(f"저자: {self.author}, 제목: {self.title}, 페이지: {self.pages}")

# 이제 책을 만들 때(인스턴스 생성 시) 바로 재료(아규먼트)를 넣어주면 돼!
book4 = Book("왕새우", "수학감각", 150) # 이 재료들이 __init__의 author, title, pages로 쏙!
book4.print_book_info()
```

코딩새싹 🌱: 와! `Book("왕새우", ...)` 이렇게 하니까 `__init__` 함수가 알아서 실행되고, `self.author` 같은 속성들이 바로 정해졌어요! 훨씬 편한데요!

<3. 우리 모두의 정보 vs. 나만의 정보 (클래스 속성 vs. 인스턴스 속성)>

선생님: `__init__` 안에서 `self.author`처럼 `self.`을 붙여서 만든 속성들은 각 인스턴스(각각의 책)만 가지는 '나만의 정보', 즉 **인스턴스 속성**이야. 그런데, 그 설계도(클래스)로 만들어진 모든 인스턴스들이 공통으로 가지는 정보도 있을 수 있어. 이걸 **클래스 속성**이라고 해.

궁금이 💡: 예를 들면, 모든 책은 '종이로 만들어졌다' 같은 공통 정보요?

선생님: 그렇지! 또는, "우리 도서관에서 이 Book 설계도로 책을 몇 권이나 만들었을까?" 하는 전체 개수를 세는 정보도 클래스 속성으로 만들 수 있어!

```
class Book:
  # 이 Book 설계도로 만들어진 모든 책들이 공유하는 정보 (클래스 속성)
  count = 0 # 처음엔 만들어진 책이 없으니 0권!

  def __init__(self, author, title, pages):
    self.author = author # 이건 인스턴스 속성
    self.title = title   # 이것도 인스턴스 속성
    self.pages = pages   # 이것도 인스턴스 속성
    Book.count += 1      # 새 책이 한 권 만들어질 때마다, "Book 설계도의" count를 1씩 증가!
    print(f"'{self.title}' 책 생성! (총 {Book.count}권째 책)")

  def print_book_info(self):
    print(f"저자: {self.author}, 제목: {self.title}, 페이지: {self.pages}")
```

```
book1 = Book("왕새우", "수학감각", 200)
print("book1을 만들고 난 후 총 책의 수:", book1.count) # 인스턴스를 통해 클래스 속성에 접근 가능!
# 또는 Book.count 로도 접근 가능! print(Book.count)

book10 = Book("이순신", "난중일기", 1000)
print("book10을 만들고 난 후 총 책의 수:", book10.count)
```

코딩새싹 🌱: Book.count += 1 이렇게 하니까 책을 만들 때마다 count가 늘어나네요! book1.count랑 book10.count가 같은 값을 보여주는 것도 신기해요! 모두가 같은 정보를 보는 거군요!

<4. 실전 연습! 멍멍이와 동그라미 만들기!>

선생님: 자, 이제 배운 걸로 귀여운 'Dog' 클래스랑 'Circle' 클래스를 만들어보자!

[멍멍이 클래스]

```
class Dog:
  species = "포유류" # 모든 강아지는 포유류 (클래스 속성)

  def __init__(self, name, breed):
    self.name = name   # 이 강아지만의 이름 (인스턴스 속성)
    self.breed = breed # 이 강아지만의 품종 (인스턴스 속성)

  def dog_intro(self): # 멍멍이가 자기소개하는 능력!
    print(f"멍멍! 내 이름은 {self.name}이고, 나는 멋진 {self.breed}야! 우리는 모두 {self.species} 친구들

puppy = Dog("puppy", "말티즈")
puppy.dog_intro()
print(f"{puppy.name}는 {puppy.species}일까? 응!")

dog2 = Dog("베이비", "불독")
dog2.dog_intro()
```

[동그라미 클래스]

```
class Circle:
  pi = 3.14 # 모든 원은 같은 원주율을 사용 (클래스 속성)

  def __init__(self, r): # 원이 만들어질 때 반지름(r)을 받는다
    self.r = r # 이 원만의 반지름 (인스턴스 속성)

  def get_circum(self): # 원의 둘레를 계산하는 능력!
    return round(self.r * 2 * Circle.pi, 2) # 인스턴스 속성(self.r)과 클래스 속성(Circle.pi)을 함께 사용!

  def get_area(self): # 원의 넓이를 계산하는 능력!
    return round(self.r ** 2 * Circle.pi, 2)

c = Circle(9)
print(f"\n반지름이 {c.r}인 원의 둘레는 {c.get_circum()}이고, 넓이는 {c.get_area()}입니다.")
```

궁금이 💡: `Circle.pi` 처럼 클래스 속성은 클래스 이름을 통해서도 접근하고, `self.r` 처럼 인스턴스 속성은 `self`를 통해서 접근하는군요! 그리고 메소드 안에서 이 둘을 섞어서 쓸 수도 있고요!

선생님: (활짝 웃으며) 바로 그거야! 오늘 우리는 클래스에 '행동'을 부여하는 메소드, 객체가 태어날 때 꼭 필요한 `__init__` 생성자, 그리고 모두가 공유하는 클래스 속성과 각자 가지는 인스턴스 속성의 차이까지 배웠어. 이제 여러분은 정말 '살아있는' 객체를 만드는 설계도를 그릴 수 있게 된 거란다! 이 정도면 파이썬 세계의 건축가라고 해도 되겠는걸? 훌륭해! 🌟