<[파이썬 수학] 제15장. 나만의 '타입' 만들기, 클래스(Class)! (새로운 탐험)>

선생님: (다양한 모양의 빵틀과 갓 구운 빵들을 다시 한번 보여주며) 얘들아, 다시 만나서 반가워! 우리가 파이썬 세상의 모든 것이 '객체'라고 이야기했었지? 그리고 `my_list.append(5)` 나 `"abc".upper()` 처럼 객체들은 각자 특별한 능력(메소드)을 가지고 있다는 것도!

코딩새싹 🌱: 네! `my_list`는 리스트 객체고, append는 리스트 객체의 능력이었어요! `"abc"`는 문자열 객체고, upper는 문자열 객체의 능력이고요!

선생님: 아주 잘 기억하고 있구나! 그럼 `type(my_list)`를 하면 `<class 'list'>` 라고 나왔던 것, `type("abc")`를 하면 `<class 'str'>` 이라고 나왔던 것은 무슨 뜻이었을까? 바로 `list`와 `str`이 이런 객체들을 만들어내는 **설계도, 즉 클래스(Class)**라는 뜻이란다!

<1. 설계도(Class)와 빵(Instance) - 다시 한번!>

선생님: 클래스는 이 '빵틀'과 같다고 했지?

- **클래스(Class):** 어떤 모양과 특징을 가진 빵을 만들지 정의해놓은 **설계도** 또는 **빵틀**. 예를 들어, "별 모양 빵틀", "동물 모양 빵틀" 같은 것들이야. 이건 추상적인 '틀' 그 자체지.
- **인스턴스(Instance):** 그 빵틀(클래스)로 실제로 찍어낸 **구체적인 빵 하나하나**. 이 빵틀로 찍어낸 별 모양 빵, 저 빵틀로 찍어낸 동물 모양 빵이 바로 인스턴스란다. 이 인스턴스가 바로 우리가 다루는 '객체'가 되는 거지!

궁금이 💡: 아하! 그러니까 파이썬에는 `list`라는 설계도, `str`이라는 설계도가 이미 있어서 우리가 그걸로 실제 리스트나 문자열 인스턴스를 만들어서 썼던 거군요!

선생님: 정확해! 그럼 이제 우리도 우리만의 새로운 빵틀, 즉 새로운 설계도(클래스)를 만들어 볼까?

<2. 가장 간단한 설계도 만들기!>

선생님: 클래스를 정의할 때는 이렇게 한단다.

```
# class 클래스이름: # 클래스 이름은 보통 영어 대문자로 시작해!
#   # 이 안에는 이 클래스로 만들 객체들의 특징(변수)과
#   # 기능(함수, 즉 메소드)들을 정의할 수 있어.


# 가장 간단한, 아무 내용도 없는 클래스를 만들어볼까?
```

```
class Sample: # 'Sample'이라는 이름의 설계도를 만들 거야!
  pass      # pass는 '아무것도 안 하고 그냥 지나가세요~'라는 뜻이야.
```

코딩새싹 🌱: pass요? 정말 아무것도 안 해요?

선생님: 응! 일단은 "Sample이라는 이름의 설계도가 있다!"라고 선언만 해두는 거지. 이제 이 텅 빈 설계도로 실제 '샘플' 인스턴스를 만들어보자!

```
x = Sample() # Sample 설계도로 x라는 실제 제품(인스턴스)을 만들었어!
print(type(x)) # x의 타입을 확인해볼까?
```

궁금이 💡: 와! <class '__main__.Sample'> 이라고 나와요! 우리가 만든 Sample이라는 새로운 타입의 객체가 정말로 만들어졌네요!

<3. 기본 특징을 가진 설계도: 클래스 속성>

선생님: 이번에는 처음부터 몇 가지 특징(정보)이 정해져 있는 빵틀을 만들어볼까? '책(Book)'이라는 설계도를 만들고, 이 설계도로 만드는 모든 책은 기본적으로 '왕새우'가 지었고, 제목은 '수학감각', 쪽수는 180쪽이라고 정해두는 거야. 이런 것들을 **클래스 속성**이라고 해.

```
class Book:
  author = "왕새우"   # 이 설계도로 만든 책은 기본 작가가 '왕새우'
  title = "수학감각"  # 기본 제목은 '수학감각'
  pages = 180       # 기본 쪽수는 180쪽
```

코딩새싹 🌱: 그럼 이 Book 설계도로 책을 만들면 다 똑같은 책이 나오겠네요?

선생님: 일단은 그렇지! 한번 세 권의 책을 찍어내 볼까?

```
book1 = Book() # 책1 만들기!
book2 = Book() # 책2 만들기!
book3 = Book() # 책3 만들기!

print("책1의 제목:", book1.title) # 책1의 제목은? -> '수학감각'
print("책2의 작가:", book2.author) # 책2의 작가는? -> '왕새우'
```

궁금이 💡: 정말 다 똑같이 나오네요! 그런데 만약 책3의 작가만 '강감찬'으로 바꾸고 싶으면 어떻게 해요?

선생님: 아주 좋은 질문이야! 바로 그게 인스턴스의 마법이지! 각 빵(인스턴스)은 빵틀(클래스)의 기본 모양을 따르지만, 우리가 원하면 각 빵에 특별한 장식을 추가할 수 있어. 이걸 **인스턴스 속성**이라고 해.

```
book3.author = '강감찬' # book3이라는 '특정 빵'에만 '강감찬'이라는 작가 스티커를 딱!


print("\nbook3의 작가는?", book3.author)  # '강감찬'으로 바뀌었지!
print("book1의 작가는 여전히?", book1.author) # '왕새우' 그대로!
print("book2의 작가도 여전히?", book2.author) # '왕새우' 그대로!


# book2의 작가도 바꿔볼까?
book2.author = "정재윤"
print("\nbook2의 작가를 '정재윤'으로 바꾼 후:", book2.author)
print("book1의 작가는 여전히?", book1.author)
```

코딩새싹 🌱: 와! book3.author = '강감찬' 하니까 book3만 작가가 바뀌고, book1이나 book2는 그대로네요! 이게 인스턴스 속성이군요! 마치 기본 빵틀 모양은 같지만, 빵마다 다른 초콜릿을 올리는 것 같아요!

선생님: 정확한 비유야! 클래스에 정의된 건 '기본값' 또는 '공통 특징'이고, 인스턴스에 직접 값을 할당하면 그 인스턴스만의 '개별 특징'이 생기는 거지.

〈4. 연습! '사람' 설계도 만들기〉

선생님: 자, 그럼 이번에는 '사람(Person)'이라는 클래스를 만들고, 이름, 나이, 아이큐를 기본 특징으로 넣어볼까?

```
class Person:
  name = "" # 기본 이름은 빈칸
  age = 0   # 기본 나이는 0살
  iq = 0    # 기본 아이큐는 0


# 'paul'이라는 사람 인스턴스를 만들어보자!
paul = Person()
print("Paul의 초기 이름:", paul.name) # 아직은 빈칸이겠지?
```

```
# 이제 Paul에게 개별 특징을 부여해주자!
paul.name = "Paul"
paul.age = 30
paul.iq = 150

print("Paul의 바뀐 이름:", paul.name)
print("Paul의 나이:", paul.age)
print("Paul의 아이큐:", paul.iq)
```

궁금이 💡: 그럼 클래스를 만들 때, 이 Person 클래스처럼 name="" 이런 식으로 기본값을 정해두는 게 '클래스 속성'이고, paul.name = "Paul" 이렇게 인스턴스를 만든 다음에 특정 값을 넣어주는 게 '인스턴스 속성'이 되는 거군요!

〈5. 설계도에 '기능'은 어디에? (메소드 살짝 맛보기)〉

선생님: (다시 한번 빵틀과 빵을 보여주며) 맞아! 우리는 오늘 '설계도(클래스)'가 객체의 '특징(속성)'을 어떻게 정의하는지 집중적으로 봤어. 그런데 우리가 처음에 이야기했던 것처럼, 객체는 특징(속성)뿐만 아니라 '기능(메소드)'도 가진다고 했지? my_list.append() 처럼 말이야.

우리 Book이나 Person 설계도에도 기능을 추가할 수 있어. 함수를 클래스 안에 정의하면 그게 바로 메소드가 된단다! 예를 들면 이렇게 말이지.

```
# class Name: # 클래스 이름은 영문 대문자로 시작!
#   # 여기에 클래스 속성들 (변수들)
#
#   def 함수1이름(self, 필요한재료1, 필요한재료2): # 메소드(클래스 안의 함수)를 정의할 때는 첫 번째 재료로
#     # 이 메소드가 수행할 기능 코드
#
#   def 함수2이름(self):
#     # 이 메소드가 수행할 다른 기능 코드
```

코딩새싹 🌱: 아! 함수 정의할 때 self라는 게 또 들어가네요!

선생님: 응, 클래스 안에서 만드는 함수, 즉 메소드는 "이 기능을 어떤 인스턴스가 사용하는가?"를 알려주기 위해 self를 첫 번째 재료로 받는 특별한 약속이 있단다. 우리가 아까 my_blue_car.speed_up(30) 했던 것처럼

말이지. `my_blue_car`가 바로 `self`가 되는 거야!

오늘은 우리가 직접 클래스라는 설계도를 만들고, 그 설계도로 인스턴스라는 실제 제품을 찍어내고, 또 각 제품에 특별한 특징을 부여하는 방법을 배웠어. `__init__` 같은 더 자세한 초기 설정이나 복잡한 메소드를 만드는 건 다음 모험에서 더 깊이 탐험해보도록 하자! 오늘 정말 중요한 첫걸음을 뗀 거야!