<[파이썬 수학] 제17장. 파이썬 마법 총정리! 나만의 프로그램 만들기!>

선생님: (커다란 마법의 책을 펼치며) 얘들아, 안녕! 오늘은 우리가 지금까지 배운 여러 가지 마법 주문들 - 변수, 연산자, 조건문, 반복문, 함수, 그리고 클래스까지! - 이 모든 것을 한데 엮어서 더욱 강력하고 실용적인 프로그램을 만드는 연습을 해볼 거야! 마치 작은 부품들을 모아 멋진 로봇을 조립하는 것처럼 말이지! 🤖✨

<1. 사용자와 대화하기: `input()` 마법 복습!>

선생님: 프로그램이 사용자와 직접 대화할 수 있다면 정말 좋겠지? 그럴 때 쓰는 마법 주문이 바로 `input()`이었어. 기억나니?

```
# input("안내 문구") 는 사용자의 입력을 '문자열'로 받아온다!
# 숫자를 원하면 int() 변신 마법을 써야 했지?
# radius_str = input("원하는 크기의 원의 반지름을 입력하세요: ")
# radius = int(radius_str)
```

코딩새싹 🌱: 네! `input()`으로 받은 건 무조건 글자 취급이니까, 숫자로 쓰고 싶으면 `int()`나 `float()`로 변신시켜야 했어요!

선생님: 아주 잘 기억하고 있구나! 그럼 이 `input()` 마법과 우리가 지난 시간에 만든 `Circle` 클래스를 합쳐서, 사용자가 원하는 크기의 원 넓이와 둘레를 구해주는 프로그램을 완성해볼까? (지난 시간 `Circle` 클래스가 있다고 가정하고)

```
# (지난 시간에 만든 Circle 클래스가 있다고 가정!)
# class Circle:
#   pi = 3.14
#   def __init__(self, r): self.r = r
#   def get_area(self): return round(self.r ** 2 * Circle.pi, 2)
#   def get_circum(self): return round(self.r * 2 * Circle.pi, 2)

# ---- 여기서부터 새로운 코드 ----
# try:
#   radius_input = input("원하는 크기의 원의 반지름을 입력하세요: ")
#   radius = int(radius_input) # 숫자로 변신!
```

```
#   my_circle = Circle(radius) # 사용자가 입력한 반지름으로 Circle 인스턴스 생성!
#   my_circle_area = my_circle.get_area()
#   my_circle_circum = my_circle.get_circum()
#   print(f"반지름이 {radius}인 원의 넓이는 {my_circle_area}\n원의 둘레는 {my_circle_circum}")
# except ValueError:
#   print("앗! 숫자를 입력해야 해요!")
```

궁금이 💡: 와! input()으로 사용자한테 값을 받고, 그 값으로 우리가 만든 클래스의 인스턴스를 만들어서 바로 사용하니까 정말 프로그램이 살아 움직이는 것 같아요! try-except로 혹시 사용자가 글자를 입력했을 때의 오류도 막아주면 더 좋겠네요! (선생님이 조용히 고개를 끄덕인다)

<2. 나만의 규칙으로 모여라! Sets 클래스 만들기>

선생님: 우리가 예전에 for문과 if문으로 합집합, 교집합을 직접 만들었던 것 기억나니? 그 로직을 이제 Sets라는 우리만의 설계도(클래스) 안에 멋지게 담아볼 거야!

```
class Sets:
  def __init__(self, a, b): # 처음 만들어질 때 두 개의 리스트(재료)를 받는다!
    self.list_a = list(set(a)) # 혹시 모를 중복 제거 및 내부 복사본으로 저장
    self.list_b = list(set(b)) # 파이썬 set으로 중복 제거 후 list로 다시 변환
    print(f"집합 준비 완료! A: {self.list_a}, B: {self.list_b}")

  def union(self): # 합집합을 구하는 능력!
    # 지난 시간에 배운 *args를 쓰면 더 좋겠지만, 여기선 두 개만!
    combined_list = self.list_a + self.list_b
    result = []
    for item in combined_list:
      if item not in result:
        result.append(item)
    result.sort() # 보기 좋게 정렬!
    return result
```

```
  def intersection(self): # 교집합을 구하는 능력!
    result = []
    for item in self.list_a:
      if item in self.list_b:
        result.append(item)
    result.sort() # 보기 좋게 정렬!
    return result

list1 = [100, 2, 3, 40, 5, 2] # list1에 중복 추가
list2 = [3, 4, 10, 20, 2]

my_set_instance = Sets(list1, list2) # Sets 설계도로 실제 집합 객체 생성!
print("두 리스트의 합집합:", my_set_instance.union())
print("두 리스트의 교집합:", my_set_instance.intersection())
```

코딩새싹 🌱: __init__에서 self.list_a = list(set(a)) 이렇게 파이썬의 진짜 set으로 한번 변환했다가 다시 list로 만드는 건 왜 그런 거예요?

선생님: 아주 좋은 질문이야! 만약 list1에 처음부터 중복된 값이 들어있었다면, 그걸 우리 Sets 클래스가 알아서 처리해주면 더 똑똑하겠지? set()으로 바꾸면 중복이 자동으로 사라지고, 그걸 다시 list()로 만들어서 우리 클래스에서 사용하는 거란다. 그리고 a와 b를 직접 저장하는 대신 list(set(a)) 처럼 복사본을 만들어 저장하면, 혹시 바깥의 list1이 나중에 바뀌더라도 우리 my_set_instance 안의 값은 영향을 받지 않아서 더 안전하단다!

<3. 번호표와 이름표를 한 번에! enumerate와 튜플 언패킹>

선생님: for 반복문으로 리스트를 돌릴 때, "이게 몇 번째 물건이지?" 하고 순서 번호가 궁금할 때가 있지? 그럴 때 일일이 cnt = 0, cnt += 1 이렇게 세는 것보다 훨씬 멋진 방법이 있어! 바로 **enumerate** 마법사!

```
fruits = ["바나나", "사과", "참외", "딸기"]

# enumerate는 (번호표, 물건) 짝꿍을 차례대로 꺼내줘!
for idx, fruit in enumerate(fruits):
```

```
  print(f"{idx}번 칸의 과일은 {fruit}입니다.")
```

궁금이 💡: `for idx, fruit in enumerate(fruits):` 이 부분, 정말 신기해요! `enumerate`가 `idx`에는 번호표를, `fruit`에는 실제 과일 이름을 알아서 넣어주는 거예요?

선생님: 바로 그거야! `enumerate`는 `(0, "바나나")`, `(1, "사과")` 이런 식으로 짝꿍(튜플)을 만들어주고, `for`문에서 `idx, fruit` 처럼 변수 두 개를 써주면 그 짝꿍이 알아서 각각의 변수에 쏙! 하고 나눠 담기는 거란다. 이걸 **튜플 언패킹(Tuple Unpacking)**이라고 해. 마치 `(10, "왕새우")`라는 짝꿍 상자를 열어서 `a`에는 `10`을, `b`에는 `"왕새우"`를 담는 것과 같지!

```
# 튜플 언패킹 예시
x = (10, "왕새우")
a, b = x # x 상자를 열어서 a와 b에 나눠 담기!
print(f"a는 {a}, b는 {b}")
```

[미션] `fruits` 리스트에서 짝수 번째 위치(인덱스)에 있는 과일만 출력해보세요! (0번째, 2번째…)

코딩새싹 🌱: `enumerate`로 번호표(`idx`)를 받아서, 그 번호표가 짝수인지 `if`문으로 검사하면 되겠어요!

```
fruits = ["바나나", "사과", "참외", "딸기", "포도"]
print("\n--- 짝수 번째 과일만! ---")
for idx, fruit in enumerate(fruits):
  if idx % 2 == 0: # 인덱스가 짝수이면
    print(f"{idx}번째 과일: {fruit}")
```

<4. 슈퍼 업그레이드! `Dog` 클래스와 `BankAccount` 클래스 만들기!>

선생님: 자, 이제 오늘 배운 모든 것을 종합해서, 아주 멋지고 복잡한 설계도(클래스)를 두 개나 만들어 볼 거야! 바로 '강아지(Dog)' 클래스와 '은행 계좌(BankAccount)' 클래스지! (선생님은 사용자가 제공한 `Dog` 클래스와 `BankAccount` 클래스의 가장 완성된 버전을 보여준다)

(Dog 클래스 설명)

선생님: 이 `Dog` 클래스를 보렴. `__init__`에서 이름, 나이, 품종, 무게를 받아서 각 강아지(인스턴스)의 특징으로 저장하지. `calc_age`처럼 사람 나이로 환산한 값을 미리 계산해서 속성으로 가질 수도 있어! 그리고 `bark`, `eat`, `run` 메소드는 강아지의 무게나 나이에 따라 다른 행동을 하도록 `if-elif-else` 조건문을 아주 잘 활용했구나! 심지어 `bark` 메소드 안에서 `self.eat()` 처럼 자기 자신의 다른 능력을 부르기도 하네!

(BankAccount 클래스 설명)

선생님: 이 BankAccount 클래스도 정말 훌륭해! 계좌번호와 초기 잔액으로 계좌를 만들고, deposit(입금)과 withdraw(출금) 능력을 가졌지. 특히 withdraw 메소드에서는 "잔액이 충분한가?" (if self.balance >= num), "출금액이 0보다 큰가?" (num > 0) 같은 중요한 조건들을 꼼꼼히 확인하고 있구나!

코딩새싹 🌱: 와! my_dog.bark() 하니까 무게에 따라서 짖는 소리가 다르고, 어떤 강아지는 밥도 달라고 하네요! my_account.withdraw(200000000) 하니까 "돈이 부족합니다." 라고 똑똑하게 알려주고요!

궁금이 💡: (BankAccount 코드의 print("남은 돈: " + self.balance) 부분을 가리키며) 선생님! 그런데 만약 돈을 입금하거나 출금했을 때, "남은 돈"을 보여주는 부분에서 self.balance가 숫자인데 문자열이랑 +로 합치려고 하면 에러가 나지 않을까요?

선생님: (눈을 반짝이며) 이야! 역시 궁금이! 정말 중요한 점을 짚어주었구나! 맞아. self.balance는 숫자이기 때문에, 문자열과 합치려면 str(self.balance) 처럼 형변환 마법을 써주거나, 우리가 배운 f-string f"남은 돈: {self.balance}" 로 써줘야 안전하겠지! (선생님은 코드를 수정하는 시늉을 한다) 이렇게 직접 코드를 실행하기 전에 문제점을 찾아내는 능력이야말로 진정한 프로그래머의 자질이란다!

선생님: (마법의 책을 덮으며) 자, 오늘 우리는 정말 많은 것을 배우고, 직접 만들어보았어. 사용자로부터 직접 값을 받는 input, 클래스 안에 이전의 로직을 담아 재사용하는 법, enumerate와 튜플 언패킹으로 더 똑똑하게 반복하는 법, 그리고 이 모든 것을 종합한 멋진 Dog 클래스와 BankAccount 클래스까지! 이제 여러분은 파이썬의 기본 마법은 물론, 자신만의 마법 세계를 창조할 수 있는 설계도를 그리는 능력까지 갖추게 되었단다! 이 "파이썬 수학" 과정을 통해 배운 모든 것들이 여러분의 앞날에 멋진 밑거름이 되기를 바란다! 정말 수고 많았어, 나의 자랑스러운 꼬마 마법사들! 🎓🎉