<[파이썬 알고리즘 탐험] 심화편 제1장: 방정식 해결사! 1차 방정식과 연립방정식 정복!>

선생님: (칠판에 'ax + b = c' 라고 쓰며) 얘들아, 안녕! 수학 시간에 이런 모양의 방정식을 풀어본 적 있지? 바로 **1차 방정식**이야! 여기서 우리는 미지의 값 x를 찾아내는 모험을 떠나게 되지. 파이썬 함수로 이 x를 구하는 해결사를 만들어볼까?

<1. 한 걸음씩 해결! 1차 방정식 풀이 함수 만들기>

선생님: 1차 방정식 ax + b = c (또는 코드에서처럼 a[0]x + a[1] = a[2])에서 x를 구하려면 어떻게 해야 할까?

코딩새싹 🌱: 음… ax = c - b 로 만들고, 그 다음에 x = (c - b) / a 이렇게 구하면 돼요!

선생님: 정확해! 그런데 만약 a가 0이라면 어떻게 될까?

궁금이 💡: 앗! a가 0이면 0으로 나눌 수 없잖아요! 그럼 `0 * x + b = c` 형태가 되는데…
만약 b랑 c가 같으면 (예: `0x + 5 = 5`), x에 어떤 수를 넣어도 항상 참이니까 해가 무수히 많아지고,
만약 b랑 c가 다르면 (예: `0x + 5 = 7`), x에 어떤 수를 넣어도 거짓이니까 해가 없게 돼요!

선생님: 이야, 궁금이가 완벽하게 분석했구나! 그 모든 경우를 고려해서 1차 방정식 해결사 함수를 만들어보자. 분수를 정확하게 다루기 위해 Fraction 마법도 쓸 거야!

```
from fractions import Fraction # 분수 마법 소환!
import numpy as np # 랜덤 숫자 마법과 배열 마법 소환!

# 1차 방정식 ax + b = c 의 해를 구하는 함수 (a,b,c를 리스트로 받음)
def solve_linear_equation(coeffs): # coeffs는 [a, b, c] 형태의 리스트
  a_coeff = coeffs[0]
  b_const = coeffs[1]
  c_const = coeffs[2]

  if a_coeff == 0: # x의 계수가 0일 때
    if b_const == c_const: # 0*x + 5 = 5 같은 경우
      print("해가 무수히 많습니다.")
      return None # 특별한 값을 반환하거나, 여기서 처리를 마침
    else: # 0*x + 5 = 7 같은 경우
      print("해가 없습니다.")
      return None
  else: # x의 계수가 0이 아닐 때
    # x = (c - b) / a
    solution = Fraction(c_const - b_const, a_coeff)
    return solution

my_equation_coeffs = [2, 3, 7] # 2x + 3 = 7
```

```
solution_x = solve_linear_equation(my_equation_coeffs)
if solution_x is not None:
  print(f"방정식 {my_equation_coeffs[0]}x + {my_equation_coeffs[1]} = {my_equation_coeffs[2]} 의 해는 x =
```

〈2. 1차 방정식 문제 자판기! LinearEq 클래스〉

선생님: 자, 이 해결사 함수를 이용해서, 무작위로 1차 방정식을 내주고 답도 알려주는 '1차 방정식 문제 자판기' 클래스를 만들어볼까?

```
class LinearEq:
  def __init__(self):
    # -20에서 19 사이의 정수 3개를 무작위로 뽑아서 방정식의 계수와 상수로 사용
    self.num_list = np.random.randint(-20, 20, 3) # [a, b, c]
    # 0 또는 1을 무작위로 뽑아서 ax + b = c 또는 ax - b = c 형태를 결정
    self.pm_selector = np.random.randint(2) # pm은 plus/minus 선택용
    print("✨ 1차 방정식 문제 자판기 준비 완료! ✨")

  def display_question(self): # display_question으로 이름 변경!
    a = self.num_list[0]
    b = self.num_list[1]
    c = self.num_list[2]

    print("\n--- 오늘의 1차 방정식 문제! ---")
    if self.pm_selector == 0: # 덧셈 형태
      print(f"{a}x + {b} = {c}")
    else: # 뺄셈 형태
      print(f"{a}x - {b} = {c}")

  def get_answer(self): # get_answer로 이름 변경!
    print("--- 정답 공개! ---")
    equation_coeffs_for_solver = list(self.num_list) # 원본 수정을 피하기 위해 복사

    if self.pm_selector == 0: # ax + b = c  형태
      # solve_linear_equation 함수는 [a, b, c] 형태를 기대함
      # 우리 num_list는 이미 [a, b, c] 형태임
      return solve_linear_equation(equation_coeffs_for_solver)
    else: # ax - b = c 형태는 ax + (-b) = c 와 같지!
      equation_coeffs_for_solver[1] *= -1 # b의 부호를 바꿔서 solver에 전달
      return solve_linear_equation(equation_coeffs_for_solver)

# 자판기 사용해보기!
```

```
problem_generator1 = LinearEq()
problem_generator1.display_question()
answer1 = problem_generator1.get_answer()
if answer1 is not None:
  print(f"정답은 x = {answer1} 입니다.")
```

코딩새싹 🌱: 와! `__init__`에서 `np.random.randint()`로 숫자들을 마구잡이로 뽑으니까 정말 자판기처럼 다른 문제가 나와요! `self.pm_selector`로 더하기 문제, 빼기 문제도 랜덤으로 나오고요!

궁금이 💡: `get_answer` 메소드에서 `else` 부분에 `equation_coeffs_for_solver[1] *= -1` 이렇게 해서 `ax - b = c`를 `ax + (-b) = c` 형태로 바꿔서 `solve_linear_equation` 함수에 똑같이 넣어주는 아이디어, 정말 똑똑한데요!

<3. 두 개의 미지수를 찾아라! 연립 1차 방정식 도전!>

선생님: (칠판에 두 개의 1차 방정식이 묶인 그림을 그리며) 자, 이번엔 미지수가 x, y 두 개이고, 식도 두 개인 **연립 1차 방정식**을 풀어볼 거야! 이건 마치 두 개의 보물 지도에 그려진 두 개의 선이 교차하는 지점(해가 되는 x, y 값)을 찾는 것과 같지!

선생님: 이 복잡한 연립방정식을 푸는 `simultaneous_eq` 함수는 넘파이 배열의 강력한 연산과 `Fraction`의 정밀함을 함께 사용한단다. (선생님은 사용자가 제공한 `simultaneous_eq` 함수와 `simultaneous_eq_print` 함수를 보여준다)

```
# (사용자가 제공한 simultaneous_eq_print 와 simultaneous_eq 함수 코드가 여기 있다고 가정)
# def simultaneous_eq_print(a, b): ...
# def simultaneous_eq(a_coeffs_list, b_coeffs_list): ...
#   a_np = np.array(a_coeffs_list, dtype=object) # Fraction을 위해 dtype=object
#   b_np = np.array(b_coeffs_list, dtype=object)
#   ... (내부 로직은 복잡하니, 핵심 아이디어 위주로 설명!)
#   핵심 아이디어: 한 변수를 소거하기 위해 한 방정식에 적절한 수를 곱한 후 다른 방정식에서 빼거나 더한다.
#   예: a[0]x + a[1]y = a[2]
#       b[0]x + b[1]y = b[2]
#       두 번째 식에 a[0]/b[0] 을 곱하면 -> a[0]x + b[1]*(a[0]/b[0])y = b[2]*(a[0]/b[0])
#       이걸 첫 번째 식에서 빼면 x가 사라지고 y에 대한 식이 나온다!
#       이 과정에서 해가 없거나 무수히 많은 경우도 판별!
```

코드가 다소 길어서, 선생님은 핵심 원리만 설명합니다:

선생님: 이 `simultaneous_eq` 함수는 마치 탐정처럼 여러 경우를 따져봐!

1. 만약 한 방정식에 x가 없다면? 바로 y를 구하고, 그걸 다른 식에 넣어서 x를 구하지.
2. 두 방정식 모두 x가 있다면? 한쪽 방정식에 똑똑한 수를 곱해서 두 방정식의 x 앞의 숫자를 똑같이 만들어! (또는 부호만 다르게)
3. 그 다음 두 식을 통째로 빼거나 더하면? x가 마법처럼 뿅! 사라지고 y만 남은 간단한 식이 나와서 y를 구할 수 있지. 이걸 **소거법**이라고 해.

4. y를 구했으면, 원래 식 아무데나 y 값을 넣어서 x도 구하면 끝!
5. 물론, 이 과정에서 두 식이 사실 똑같은 식이라 해가 무수히 많거나, 두 식이 평행선처럼 절대 만나지 않아 해가 없는 경우도 꼼꼼하게 찾아낸단다!

〈4. 연립방정식 문제 자판기! `SimultaneousEq` 클래스〉

선생님: 자, 이 똑똑한 simultaneous_eq 해결사 함수를 이용해서, 이번엔 '연립방정식 문제 자판기'를 만들어볼까?

```
class SimultaneousEq:
  def __init__(self):
    # -10에서 9 사이의 정수 6개를 무작위로 뽑아 두 방정식의 계수와 상수로!
    # [a1, b1, c1, a2, b2, c2]
    self.coeffs_all = np.random.randint(-9, 10, 6) # 분모/계수가 0이 되는 경우를 줄이기 위해 범위 조절
    # 계수가 0이 되면 문제가 너무 단순해지거나 해를 구하기 곤란할 수 있으므로,
    # 실제로는 0이 안 나오도록 처리하는 로직이 더 필요할 수 있어!
    if self.coeffs_all[0] == 0: self.coeffs_all[0] = 1 # ax의 a가 0이 안되게
    if self.coeffs_all[1] == 0: self.coeffs_all[1] = 1 # by의 b가 0이 안되게 (첫번째 식)
    if self.coeffs_all[3] == 0: self.coeffs_all[3] = 1 # dx의 d가 0이 안되게
    if self.coeffs_all[4] == 0: self.coeffs_all[4] = 1 # ey의 e가 0이 안되게 (두번째 식)

    print("✨ 연립방정식 문제 자판기 준비 완료! ✨")

  def display_question(self):
    c = self.coeffs_all
    print("\n--- 다음 연립방정식을 푸시오 ---")
    print(f"{c[0]}x + {c[1]}y = {c[2]}")
    print(f"{c[3]}x + {c[4]}y = {c[5]}")

  def get_answer(self):
    c = self.coeffs_all
    # simultaneous_eq 함수는 두 개의 리스트 [a1,b1,c1], [a2,b2,c2] 를 받음
    eq1_coeffs = [c[0], c[1], c[2]]
    eq2_coeffs = [c[3], c[4], c[5]]

    # 여기서 simultaneous_eq 함수를 직접 호출해야 함 (위에서 정의했다고 가정)
    # 예를 들어, 아래와 같이 호출 (실제 simultaneous_eq 함수 정의 필요)
    # x, y = simultaneous_eq(eq1_coeffs, eq2_coeffs)
    # 여기서는 일단 결과가 있다고 가정하고 출력만
    print("--- 정답 (계산 로직은 simultaneous_eq 함수에!) ---")
    # print(f"x = {x}, y = {y}") # 실제 x,y 값을 simultaneous_eq로 계산해서 넣어야 함
```

```
    # 지금은 simultaneous_eq 함수가 이 코드 블록에 없으므로, 가상의 답을 출력하거나 주석처리
    print("(simultaneous_eq 함수를 호출하여 실제 답을 계산해야 합니다)")


# 자판기 사용해보기!
# (실제 실행을 위해서는 위에 정의된 simultaneous_eq 함수가 필요합니다)
sim_problem_generator = SimultaneousEq()
sim_problem_generator.display_question()
sim_problem_generator.get_answer() # 이 부분은 simultaneous_eq 함수가 있어야 제대로 동작합니다.
```

코딩새싹 🌱: 와! 이제 연립방정식 문제도 마구마구 만들어낼 수 있겠어요! 클래스 안에 우리가 만든 함수를 부품처럼 사용하니까 정말 편리해요!

궁금이 💡: `__init__`에서 계수들을 무작위로 뽑을 때, 혹시 x나 y의 계수가 0이 되어버리면 연립방정식이 아니게 되거나 풀기 어려운 경우가 생길 수도 있겠네요! 그런 부분도 꼼꼼하게 처리하면 더 완벽한 자판기가 될 것 같아요! (선생님이 `SimultaneousEq` 클래스의 `__init__` 부분을 조금 수정한다)

선생님: (만족스러운 표정으로) 바로 그거야! 오늘 우리는 1차 방정식부터 시작해서, 조금 더 복잡한 연립 1차 방정식까지 파이썬으로 풀어내는 방법을 탐구했어. 특히, 다양한 경우의 수를 생각하고, 그것을 코드로 논리정연하게 표현하는 과정이 바로 알고리즘의 핵심이란다! 그리고 `Fraction`으로 정확한 답을, `numpy`로 편리한 계산을, 클래스로 멋진 '문제 자판기'까지 만들었으니, 여러분은 이제 진정한 '방정식 해결 마법사'라고 할 수 있겠는걸! "파이썬 알고리즘 탐험" 심화편의 첫 장부터 정말 대단한 활약이었어! 🌟