<[파이썬 고급 마법학교] 제1장: 리스트 내포, 람다 필터, 그리고 부등식 해결사!>

선생님: (마법 주문이 적힌 두루마리를 펼치며) 얘들아, 안녕! 우리가 for 반복문과 if 조건문을 써서 새로운 리스트를 만들곤 했지? 예를 들어, "1부터 29까지의 숫자 중에서, 30 * 숫자 + 60 이 600보다 작은 경우에만 그 숫자를 모아줘!" 같은 마법 말이야.

```
# for 반복문으로 조건에 맞는 숫자 찾기 (복습)
x_range = range(1, 30)
a_list_for = []
for i in x_range:
  if 30 * i + 60 < 600:
    a_list_for.append(i)
print("for문으로 찾은 숫자들:", a_list_for)
```

선생님: 그런데 파이썬 마법사들은 이런 작업을 훨씬 더 짧고 우아하게 표현하는 비밀 주문을 알고 있단다! 바로 **리스트 내포(List Comprehension)**라는 마법이지!

<1. 번개처럼 빠르게! 리스트 내포 마법!>

리스트 내포 기본 주문: [결과표현식 for 항목변수 in 데이터_묶음 if 조건문]

```
# 리스트 내포로 똑같은 작업하기!
my_list_comprehension = [i for i in range(1, 30) if 30 * i + 60 < 600]
print("리스트 내포로 찾은 숫자들:", my_list_comprehension)
```

코딩새싹 🌱: 와! for문이랑 if문이 한 줄에 쏙 들어갔어요! 정말 간결하고 멋진데요! "1부터 29까지의 i에 대해서, 만약 30 * i + 60 < 600 이라는 조건이 참이면, 그 i를 리스트에 넣어줘!" 라고 바로 읽혀요!

[퀴즈] 1부터 10까지 숫자 중 2의 배수만 골라서 리스트를 만들어보세요! (두 가지 방법으로!)

```
# 방법 1: if 조건으로 2의 배수 찾기
even_list_v1 = [i for i in range(1, 11) if i % 2 == 0]
print("\n2의 배수 리스트 (방법1):", even_list_v1)

# 방법 2: 2를 곱해서 2의 배수 만들기 (1부터 5까지의 숫자에 2를 곱하면?)
even_list_v2 = [i * 2 for i in range(1, 6)] # 1*2, 2*2, 3*2, 4*2, 5*2
print("2의 배수 리스트 (방법2):", even_list_v2)
```

[생활 속 문제 해결!]

> 한 권에 2000원 하는 공책이 할인 마트에서는 1200원에 팔린대. 그런데 할인 마트까지 다녀오는데 왕복 교통비가 2400원이 든다면, 공책을 몇 권 이상 살 때 할인 마트에서 사는 게 더 이득일까?

궁금이 💡: 음… 리스트 내포로 "할인 마트에서 사는 게 더 싼 공책 권수"를 다 찾아볼 수 있겠어요! (2000 * i)랑 (1200 * i + 2400)을 비교해서요!

```
# 할인 마트가 더 싼 모든 경우의 공책 권수 (1권부터 49권까지 시도)
# 조건: (원래 가격 * 권수) > (할인 가격 * 권수 + 교통비)
```

```
favorable_notebook_counts = [i for i in range(1, 50) if 2000 * i > 1200 * i + 2400]
print("\n할인 마트가 더 이득인 공책 권수들:", favorable_notebook_counts)
# 그런데 우리가 진짜 궁금한 건 "몇 권 '이상'부터" 이득인가? 하는 딱 그 지점이지!
# 이럴 땐 직접 방정식을 푸는 게 더 정확할 수도 있어!
# 2000 * x = 1200 * x + 2400  => 800 * x = 2400 => x = 3
# 즉, 3권일 때는 가격이 같고, 3권보다 "많이" 사야 이득이 시작돼!
original_price = 2000
discount_price = 1200
transportation_fee = 2400 # 문제에서는 2400원이었네요!
breaking_point = transportation_fee / (original_price - discount_price) # 2400 / 800 = 3
print(f"딱 같아지는 지점: {breaking_point}권")
print(f"정답: {int(breaking_point) + 1}권 이상부터 할인 마트가 유리!")
```

선생님: 아주 좋아! 리스트 내포는 가능한 모든 경우를 탐색할 때 유용하고, 때로는 이렇게 직접 수학 공식을 푸는 게 더 명쾌한 답을 주기도 한단다!

<2. 한 줄 마법 lambda와 마법의 체 filter 복습!>

선생님: 우리가 lambda로 간단한 함수를 만들고, filter로 조건에 맞는 것만 골라냈던 것 기억나지?

```
import numpy as np # 넘파이 마법사도 함께!

# lambda 복습: 제곱 함수
square_lambda = lambda x: x ** 2
print(f"\n4의 제곱 (lambda): {square_lambda(4)}")

# lambda와 numpy 배열의 만남!
x_np_array = np.arange(-10, 11) # -10부터 10까지의 넘파이 배열
# 각 숫자에 5를 더한 값이 4보다 크거나 같은지 True/False로 알려줘!
is_condition_met_lambda = lambda a: a + 5 >= 4
print("x_np_array에 lambda 적용 결과:", is_condition_met_lambda(x_np_array))
# (참고: 일반 파이썬 리스트에는 이런 식의 한 번에 적용은 안 된단다!)

# filter 복습: 0부터 10까지 숫자 중 짝수만 골라내기
numbers_range = range(1, 11)
even_numbers_filtered = list(filter(lambda x: x % 2 == 0, numbers_range))
print("filter로 찾은 짝수들:", even_numbers_filtered)
```

<3. 부등식 문제 해결사, Inequality 클래스 만들기!>

선생님: 자, 이제 이 모든 마법을 합쳐서, 무작위로 1차 부등식 문제를 내고 그 해까지 구해주는 '부등식 문제 해결사' 로봇을 만들어보자! ax + b > c 이런 형태의 문제 말이야!

```
class InequalitySolver:
```

```
def __init__(self):
  # 문제에 사용할 숫자 a, b, c를 -10에서 9 사이에서 무작위로 뽑는다.
  self.coeffs = np.random.randint(-9, 10, 3) # a, b, c (a가 0이 안 되게!)
  while self.coeffs[0] == 0: # 만약 x의 계수 a가 0이면 다시 뽑자!
    self.coeffs[0] = np.random.randint(-9, 10)
    if self.coeffs[0] == 0: self.coeffs[0] = 1 # 그래도 0이면 1로 강제!

  self.operator_type = np.random.randint(2) # 0이면 +, 1이면 -
  self.inequality_type = np.random.randint(4) # 0:>, 1:>=, 2:<, 3:<=
  print("✨ 1차 부등식 문제 해결사 등장! ✨")

def display_question(self):
  a, b, c = self.coeffs[0], self.coeffs[1], self.coeffs[2]

  op_symbol = "+" if self.operator_type == 0 else "-"

  ineq_symbols = [">", ">=", "<", "<="]
  ineq_symbol = ineq_symbols[self.inequality_type]

  print(f"\n--- 오늘의 부등식 문제 (x는 -10부터 10까지 정수) ---")
  print(f"{a}x {op_symbol} {b} {ineq_symbol} {c}")

def find_solutions(self): # find_solutions로 이름 변경!
  a, b_orig, c = self.coeffs[0], self.coeffs[1], self.coeffs[2]

  # 만약 문제가 "ax - b" 형태였다면, b의 부호를 바꿔서 "ax + (-b)" 형태로 통일!
  b_eff = b_orig if self.operator_type == 0 else -b_orig

  # x의 범위는 -10부터 10까지의 정수로 한정해서 찾아보자.
  possible_x_values = range(-10, 11)

  solutions = []
  # 선택된 부등호에 따라 올바른 lambda 함수를 filter에 넘겨준다!
  if self.inequality_type == 0: # >
    solutions = list(filter(lambda x: a * x + b_eff > c, possible_x_values))
  elif self.inequality_type == 1: # >=
    solutions = list(filter(lambda x: a * x + b_eff >= c, possible_x_values))
  elif self.inequality_type == 2: # <
    solutions = list(filter(lambda x: a * x + b_eff < c, possible_x_values))
```

```
    else: # <=
      solutions = list(filter(lambda x: a * x + b_eff <= c, possible_x_values))

    print("--- 정답 ---")
    return solutions

# 해결사 로봇 사용해보기!
ineq_solver = InequalitySolver()
ineq_solver.display_question()
print("해(x)가 될 수 있는 정수들:", ineq_solver.find_solutions())
```

코딩새싹 🌱: 와! `__init__`에서 `while self.coeffs[0] == 0:` 이렇게 해서 x의 계수가 0이 되지 않도록 막아주는 거 정말 똑똑해요!

궁금이 💡: `find_solutions` 메소드에서 `if-elif-else`로 각 부등호에 맞는 `lambda` 함수를 `filter`에 쏙 넣어주니까, 어떤 부등식 문제든 다 풀 수 있겠네요!

<4. 업그레이드! 연립 부등식 해결사, `SimIneq` 클래스!>

선생님: 하나의 부등식을 푸는 것도 멋지지만, 두 개의 부등식을 동시에 만족하는 해를 찾는 '연립 부등식' 문제는 어떨까? 우리가 방금 만든 `InequalitySolver`의 아이디어를 확장하면 충분히 만들 수 있어!

```
# 먼저, 하나의 부등식 해를 구하는 헬퍼 함수를 만들자 (InequalitySolver의 find_solutions와 유사)
def solve_single_inequality(coeffs, operator_type, inequality_type, x_range=range(-10, 11)):
    a, b_orig, c = coeffs[0], coeffs[1], coeffs[2]
    b_eff = b_orig if operator_type == 0 else -b_orig

    if inequality_type == 0: return list(filter(lambda x: a * x + b_eff > c, x_range))
    elif inequality_type == 1: return list(filter(lambda x: a * x + b_eff >= c, x_range))
    elif inequality_type == 2: return list(filter(lambda x: a * x + b_eff < c, x_range))
    else: return list(filter(lambda x: a * x + b_eff <= c, x_range))

class SimultaneousInequalities:
  def __init__(self):
    self.coeffs1 = np.random.randint(-9, 10, 3); self.op_type1 = np.random.randint(2); self.ineq_type1 =
    self.coeffs2 = np.random.randint(-9, 10, 3); self.op_type2 = np.random.randint(2); self.ineq_type2 =
    # x의 계수가 0이 안 되도록 보장 (간단화)
    while self.coeffs1[0] == 0: self.coeffs1[0] = np.random.randint(1,10) * random.choice([-1,1])
    while self.coeffs2[0] == 0: self.coeffs2[0] = np.random.randint(1,10) * random.choice([-1,1])
    print("✨ 연립 부등식 문제 해결사 등장! ✨")

  def display_question(self):
```

```
        print("\n--- 다음 연립 부등식을 푸시오 (x는 -10부터 10까지 정수) ---")
        s = [">", ">=", "<", "<="] # 부등호 목록
        op_s = ["+", "-"] # 연산자 목록
        print(f"{self.coeffs1[0]}x {op_s[self.op_type1]} {abs(self.coeffs1[1])} {s[self.ineq_type1]} {self.co
        print(f"{self.coeffs2[0]}x {op_s[self.op_type2]} {abs(self.coeffs2[1])} {s[self.ineq_type2]} {self.co

    def find_common_solutions(self): # find_common_solutions로 이름 변경!
        solutions1 = solve_single_inequality(self.coeffs1, self.op_type1, self.ineq_type1)
        solutions2 = solve_single_inequality(self.coeffs2, self.op_type2, self.ineq_type2)

        print(f"첫 번째 부등식의 해: {solutions1}")
        print(f"두 번째 부등식의 해: {solutions2}")

        # 두 해 리스트의 공통된 원소(교집합)를 찾는다!
        common_sols = list(filter(lambda x: x in solutions2, solutions1))
        print("--- 정답 ---")
        return common_sols

# 연립 부등식 해결사 사용해보기!
sim_ineq_solver = SimultaneousInequalities()
sim_ineq_solver.display_question()
print("두 부등식을 모두 만족하는 x:", sim_ineq_solver.find_common_solutions())
```

코딩새싹 🌱: 와! `solve_single_inequality`라는 함수를 따로 만들어두니까, `SimultaneousInequalities` 클래스 안에서 그 함수를 두 번 불러서 각 부등식의 해를 구하고, 그 다음에 `filter`로 공통된 해를 찾는군요! 함수로 만드니 코드가 훨씬 깔끔하고 재사용하기 좋아요!

궁금이 💡: `np.random.choice(signs, 2)` 이런 식으로 `signs` 리스트에서 2개를 무작위로 뽑을 수도 있군요! 그리고 `self.sign1, self.sign2 = ...` 이렇게 변수 두 개에 한 번에 나눠 담는 건 '언패킹'이라고 하셨죠!

선생님: (활짝 웃으며) 맞아! 오늘 우리는 리스트 내포로 코드를 간결하게 만들고, 람다와 필터로 데이터를 입맛대로 요리하고, 이 모든 것을 클래스라는 멋진 로봇 안에 담아 부등식 문제 해결사까지 만들어봤어! 이제 여러분은 파이썬의 고급 마법들을 자유자재로 구사하며 어떤 문제든 자신있게 해결할 수 있는 '고급 마법사'가 되었단다! 정말 자랑스러워! 🥳