<[파이썬 알고리즘 탐험] 응용편: 숫자 마법과 문제 해결사 클래스!>

선생님: (여러 가지 모양의 숫자 블록들을 보여주며) 얘들아, 안녕! 오늘은 흩어져 있는 숫자 조각들을 모아 하나의 큰 숫자로 만들거나, 특별한 규칙을 가진 숫자들의 비밀을 파헤치고, 심지어 우리만의 '정수 문제 해결사' 로봇까지 만들어 볼 거야!

<1. 흩어진 숫자 구슬 꿰기! - 리스트를 숫자로 변환하기>

선생님: 만약 우리에게 `[5, 2, 3, 4]` 처럼 각 자릿수가 담긴 리스트가 있다면, 이걸 어떻게 5234라는 하나의 숫자로 합칠 수 있을까?

코딩새싹 🌱: 으음… 각 숫자를 글자로 바꿔서 쭉~ 이어 붙인 다음에, 마지막에 다시 숫자로 변신시키면 되지 않을까요?

선생님: 아주 좋은 생각이야! 그게 바로 첫 번째 방법이지!

방법 1: 글자 기차로 변신 후 숫자 재탄생!

```
def list_to_num_string_conversion(mylist):
  num_string = "" # 빈 글자 기차 준비!
  for i in mylist:
    num_string += str(i) # 각 숫자를 글자로 바꿔서 기차에 연결!
  return int(num_string) # 완성된 글자 기차를 다시 숫자로 변신!

a = [5, 2, 3, 4]
print(f"{a} 리스트가 숫자로 변신 (방법1): {list_to_num_string_conversion(a)}")
```

궁금이 💡: 그럼 수학적인 방법은 없을까요? 각 숫자의 자릿값을 이용하는 거예요! 5234는 `5*1000 + 2*100 + 3*10 + 4*1` 이잖아요!

선생님: 이야, 궁금이는 역시 핵심을 꿰뚫는구나! 바로 그 원리를 이용한 두 번째 방법도 있단다!

방법 2: 자릿값의 마법으로 숫자 조립!

```
def list_to_num_math(mylist):
  num = 0
  list_length = len(mylist)
  for i, val in enumerate(mylist): # 번호표(i)와 값(val)을 함께 받자!
    # val 에다가 10의 (전체길이 - 현재번호표 - 1) 제곱을 곱해준다!
    # 예: [5,2,3,4] 에서 5(i=0)는 10^(4-0-1)=10^3, 2(i=1)는 10^(4-1-1)=10^2 ...
    power = list_length - i - 1
    num += val * (10 ** power)
  return num

print(f"{a} 리스트가 숫자로 변신 (방법2): {list_to_num_math(a)}")
```

코딩새싹 🌱: 와! 두 가지 방법 모두 같은 결과가 나왔어요! 두 번째 방법은 정말 수학적 원리를 이용하는 느낌이네요!

<2. 토끼들의 비밀 암호? 피보나치 수열!>

선생님: (토끼 그림을 보여주며) 혹시 피보나치 수열이라고 들어봤니? 어떤 이탈리아 수학자가 토끼가 얼마나 빨리 번식하는지 연구하다가 발견한 아주 신기한 숫자 규칙이란다. "앞의 두 숫자를 더하면 다음 숫자가 되는" 마법 같은 수열이지! 보통 1, 1로 시작해서 1, 1, 2, 3, 5, 8, 13, 21… 이렇게 쭉~ 이어진단다.

궁금이 💡: 그럼 n번째 피보나치 숫자를 찾아내는 함수도 만들 수 있겠네요!

선생님: 물론! 두 가지 방법으로 만들어보자!

방법 1: 리스트에 차곡차곡 쌓아가기!

```
def fibonacci_list(n):
  if n <= 0: return "n은 1 이상이어야 해요!"
  if n == 1 or n == 2: return 1 # 첫 번째와 두 번째는 1

  my_list = [1, 1] # 처음 두 숫자로 시작!
  # n-1번 반복하면 n개의 숫자가 리스트에 담기는데, 이미 2개가 있으니 n-2번만 더 반복!
  # range(1, n-1)은 n-2번 반복 (예: n=6이면 range(1,5) -> i는 1,2,3,4)
  for i in range(1, n - 1):
    # 리스트에서 i번째 숫자와 (i-1)번째 숫자를 더해서 다음 숫자를 만든다.
    # 주의: 여기서 i는 반복 횟수를 세는 용도, 리스트 인덱스로는 현재 리스트의 길이를 활용해야 해
    # 또는, 리스트의 마지막 두 숫자를 더하는 방식으로!
    next_num = my_list[-2] + my_list[-1] # 더 직관적인 방법: 리스트의 마지막 두 개를 더한다!
    my_list.append(next_num)
  return my_list[-1] # 최종적으로 만들어진 리스트의 마지막 숫자가 n번째 피보나치 수!

print(f"\n피보나치 수열 (리스트 방식) 6번째 수는? {fibonacci_list(6)}")
```

코딩새싹 🌱: `my_list[-2] + my_list[-1]` 이렇게 리스트의 맨 뒤 두 개를 더하니까 바로 다음 숫자가 나오는군요!

방법 2: 두 개의 바구니만으로 똑똑하게!

```
def fibo_vars(n):
  if n <= 0: return "n은 1 이상이어야 해요!"
  a, b = 0, 1 # 첫 두 수를 0, 1로 시작해서 (또는 1,1로 시작할 수도 있음)
              # n번째 수를 찾을 때, 1,1,2,3,5,8... 이렇게 b가 그 수를 가리키게 한다.
              # 혹은 a가 그 수를 가리키게 할 수도 있다. (아래 코드는 a가 n번째 피보나치)
  if n == 1: return b # 1번째는 1

  # 피보나치 수열의 정의에 따라 F(0)=0, F(1)=1, F(2)=1, F(3)=2 ...
```

```
  # 아래 코드는 n번째 피보나치 수를 a에 담아 반환한다. (단, F(0)=0, F(1)=1 기준 n)
  # 만약 F(1)=1, F(2)=1 기준으로 n번째를 원한다면, for i in range(n-1) 또는 초기값 조정이 필요.
  # 제공된 코드는 1,1,2,3,5,8... 에서 6번째 값인 8을 반환하므로,
  # F(1)=1, F(2)=1 기준.
  a, b = 1, 1 # 첫번째와 두번째를 1로 초기화 (F(1), F(2))
  if n == 1 or n == 2: return 1

  for _ in range(n - 2): # F(3)부터 F(n)까지 구하려면 n-2번 반복
    a, b = b, a + b # a는 이전 b가 되고, b는 이전 a와 이전 b의 합이 된다! (마법의 교환!)
  return b # (n-2)번 반복 후 b가 n번째 피보나치 수가 됨

print(f"피보나치 수열 (변수 교환 방식) 6번째 수는? {fibo_vars(6)}")
```

궁금이 💡: a, b = b, a + b 이 한 줄이 정말 마법 같아요! a에는 이전 b의 값을, b에는 이전 a와 b를 더한 새로운 값을 동시에 착착 넣어주네요! 리스트를 안 쓰고도 할 수 있다니 신기해요!

<3. 업그레이드! 정수 문제 해결사 로봇 만들기!>

선생님: 자, 이제 우리가 지금까지 배운 모든 지혜와 넘파이의 랜덤 마법까지 합쳐서, 스스로 정수 연산 문제를 내고 답까지 알려주는 '정수 문제 해결사 로봇' 클래스를 만들어볼까?

```
import numpy as np # 넘파이의 랜덤 기능을 사용하기 위해!

class IntegerProblemSolver: # 클래스 이름은 대문자로 시작!

  def __init__(self): # 로봇이 만들어질 때 실행되는 초기 설정!
    # 문제에 사용할 두 개의 숫자를 -10 ~ 9 사이에서 무작위로 뽑는다.
    self.constants = np.random.randint(-10, 10, size=2) # size=2 로 수정!
    # 어떤 연산을 할지 0~4 사이에서 무작위로 선택! (0:덧셈, 1:뺄셈, 2:곱셈, 3:나눗셈, 4:지수)
    self.operation_type = np.random.randint(5)
    # 지수 연산에 사용할 지수를 2~3 사이에서 무작위로 선택!
    self.exponent = np.random.randint(2, 4)
    print("🤖 정수 문제 해결사 로봇 등장! 어떤 문제가 나올까?")

  def ask_question(self): # 문제를 내주는 능력!
    num1 = self.constants[0]
    num2 = self.constants[1]

    print("\n--- 문제---")
    if self.operation_type == 0: # 덧셈 문제
      print(f"{num1} + {num2} = ?")
    elif self.operation_type == 1: # 뺄셈 문제
```

```
      print(f"{num1} - {num2} = ?")
    elif self.operation_type == 2: # 곱셈 문제
      print(f"{num1} * {num2} = ?")
    elif self.operation_type == 3: # 나눗셈 문제
      # 주의: 0으로 나누는 경우를 생각해줘야 하지만, 여기선 일단 생략!
      print(f"{num1} / {num2} = ? (결과는 실수로 나타날 수 있어요!)")
    else: # 지수 문제
      print(f"{num1} ^ {self.exponent} = ?")

  def tell_answer(self): # 정답을 알려주는 능력!
    num1 = self.constants[0]
    num2 = self.constants[1]
    answer = 0

    print("---정답---")
    if self.operation_type == 0:
      answer = num1 + num2
    elif self.operation_type == 1:
      answer = num1 - num2
    elif self.operation_type == 2:
      answer = num1 * num2
    elif self.operation_type == 3:
      if num2 == 0: # 0으로 나누는 경우!
          return "0으로는 나눌 수 없어요!"
      answer = num1 / num2
    else:
      answer = num1 ** self.exponent
    return answer

# 로봇 한 대를 만들어볼까?
problem_robot1 = IntegerProblemSolver()
problem_robot1.ask_question() # 로봇아, 문제 내줘!
print(problem_robot1.tell_answer()) # 로봇아, 답도 알려줘!

print("\n--- 새로운 문제! ---")
problem_robot2 = IntegerProblemSolver() # 새 로봇은 새 문제를 내겠지?
problem_robot2.ask_question()
print(problem_robot2.tell_answer())
```

코딩새싹 🌱: 와! `IntegerProblemSolver()` 이렇게 로봇을 만들 때마다 `__init__` 덕분에 다른 숫자랑 다른

종류의 문제가 준비되네요! 그리고 ask_question()이랑 tell_answer() 메소드를 부르면 문제도 내주고 답도 알려주고요!

궁금이 💡: self.constants에 np.random.randint(-10, 10, size=2) 이렇게 해서 숫자 두 개를 한 번에 뽑는 것도 넘파이의 편리한 기능이군요! tell_answer() 메소드가 숫자를 return 해주니까 print()로 그 결과를 바로 볼 수 있고요! 만약 나누기 문제에서 num2가 0이 되는 경우도 친절하게 알려주면 더 완벽한 로봇이 될 것 같아요! (선생님이 tell_answer 함수에 해당 로직을 추가한다)

선생님: (만족스러운 표정으로) 아주 훌륭해! 오늘 우리는 숫자 리스트를 하나의 숫자로 합치는 방법, 신비로운 피보나치 수열을 만드는 두 가지 방법, 그리고 이 모든 지혜를 모아 스스로 문제를 내고 푸는 '정수 문제 해결사 로봇'까지 만들어봤어! 이렇게 작은 아이디어와 코드 조각들이 모여서 점점 더 복잡하고 유용한 프로그램을 만들어낼 수 있다는 것, 정말 멋지지 않니? 여러분의 알고리즘 탐험은 이제부터가 진짜 시작이야! 🚀