<[파이썬 데이터 마법사] 제2장: 넘파이 배열의 변신! 모양과 조건의 마법!>

선생님: (여러 모양의 블록들을 보여주며) 얘들아, 안녕! 우리가 프로그래밍을 하다 보면 "이 조건도 괜찮고, 저 조건도 괜찮아!" 처럼 여러 조건 중 하나만 만족해도 되는 경우가 있어. 이럴 때 쓰는 논리 마법이 바로 **or** 란다!

<1. "이거나 저거거나!" - 논리 연산자 or>

```
a = 3
b = 10

# a가 4보다 크거나 "또는" b가 5보다 크면 True!
print(f"a > 4 or b > 5 의 결과는? {a > 4 or b > 5}") # a > 4는 False, b > 5는 True. or는 둘 중 하나만 Tru
```

코딩새싹 🌱: 아! or는 양쪽 조건 중에 하나라도 True면 전체가 True가 되는 거군요! 자판기에 500원짜리 동전을 넣거나 "또는" 1000원짜리 지폐를 넣으면 음료수가 나오는 거랑 비슷해요!

선생님: 정확한 비유야! 그럼 이 or 마법을 써서 1부터 30까지 숫자 중에서 2의 배수이거나 또는 3의 배수인 숫자들을 찾아볼까?

```
c = range(1, 31) # 1부터 30까지의 숫자들이 담긴 range 객체

print("\n--- 2의 배수 또는 3의 배수 (for문) ---")
for i in c:
  if i % 2 == 0 or i % 3 == 0: # i가 2로 나누어떨어지거나 또는 3으로 나누어떨어지면,
    print(i, end=" ") # 출력하고 한 칸 띄기

# 이걸 리스트 내포로도 할 수 있겠지?
print("\n\n--- 2의 배수 또는 3의 배수 (리스트 내포) ---")
multiples_of_2_or_3 = [i for i in c if i % 2 == 0 or i % 3 == 0]
print(multiples_of_2_or_3)
```

<2. 내 배열은 어떤 모양? - .shape 과 .reshape()>

선생님: 우리가 만든 넘파이 배열은 사실 '모양(shape)'이라는 비밀 정보를 가지고 있단다!

```
import numpy as np # 넘파이 마법사 다시 소환!

a = [1, 2, 3, 4, 5, 6]
b_np = np.array(a) # 넘파이 배열로 변신!

print("\n--- 넘파이 배열의 모양 ---")
print("b_np 배열:", b_np)
print("b_np의 모양은?", b_np.shape) # (6,) 이라고 나오네! 이건 "6개의 원소를 가진 1차원 배열"이라는 뜻이
```

궁금이 💡: (6,) 뒤에 쉼표는 왜 있는 거예요?

선생님: 좋은 질문이야! 넘파이는 다차원 배열을 다룰 수 있어서, (6,)은 "첫 번째 차원에 6개의 원소가 있다"는 것을 명확히 알려주기 위한 표현이란다. 자, 그럼 이 1차원 배열을 마치 로봇처럼 다른 모양으로 변신시켜 볼까? **.reshape(행, 열)** 마법을 쓰면 돼!

```
# b_np (6개 원소)를 2행 3열짜리 모양으로 변신! (2 * 3 = 6 이니까 가능!)
reshaped_b = b_np.reshape(2, 3)
print("모양 변신 후 (2행 3열):")
print(reshaped_b)
print("변신 후 모양은?", reshaped_b.shape) # (2, 3) 이라고 나오지!
```

코딩새싹 🌱: 와! 일렬로 서 있던 병정 6명을 2줄 3칸으로 딱 맞춰서 세운 것 같아요! 그럼 원래 숫자 개수랑 안 맞게 변신시킬 수도 있어요?

선생님: 그건 안 된단다! reshape 마법은 원래 있던 원소의 총 개수가 변신하려는 모양의 총 개수와 똑같아야만 성공할 수 있어!

〈3. 2차원 배열의 세계로! (행렬 맛보기)〉

선생님: 이제 숫자들이 표처럼 가로세로로 놓인 **2차원 배열**을 만들어보자! 마치 엑셀 시트나 게임의 지도 같지.

```
c_np = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 리스트 안에 리스트를 넣어서 2차원 배열 만들기!
print("\n--- 2차원 배열 c_np ---")
print(c_np)
```

선생님: 2차원 배열에서 특정 위치의 숫자를 꺼내려면, 마치 영화관 좌석을 찾듯이 '몇 번째 줄(행), 몇 번째 칸(열)'인지 알려줘야 해! (컴퓨터는 0번 줄, 0번 칸부터 센다는 것 잊지 않았지?)

```
# c_np[행번호, 열번호]
print("1행 2열의 값은? (0부터 세서):", c_np[1, 2]) # 6

# 슬라이싱도 가능해!
print("1행의, 0번 칸부터 2번 칸 '전'까지:", c_np[1, 0:2]) # [4 5]
print("1행부터 끝까지, 그 중 1번 칸만:", c_np[1:, 1])    # [5 8] (1행 1열, 2행 1열)
print("맨 뒤 행, 맨 뒤 열의 값:", c_np[-1, -1])         # 9
```

궁금이 💡: 쉼표(,)로 행이랑 열을 구분해서 인덱싱하니까 정말 좌표 찾는 것 같아요! 슬라이싱도 각 차원별로 적용할 수 있군요!

[도전!] 1부터 25까지의 숫자로 5행 5열짜리 넘파이 배열을 만들어보세요!

```
a_25 = np.array(range(1, 26)) # 1부터 25까지 숫자를 만들고,
reshaped_a_25 = a_25.reshape(5, 5) # 5행 5열로 변신!
print("\n--- 5x5 배열 ---")
print(reshaped_a_25)
```

〈4. 빈 도화지 준비! - zeros 와 ones〉

선생님: 때로는 계산을 시작하기 전에 특정 크기의 배열을 만들어두고, 모든 값을 0이나 1로 채워놓고 싶을 때가 있어. 마치 그림 그리기 전에 하얀 도화지(zeros)를 준비하거나, 특정 밑바탕색(ones)으로 한번 칠해놓는 것과 같지.

```
# 모든 값이 0으로 채워진 2행 4열 배열 만들기
zeros_array = np.zeros((2, 4))
print("\n--- 0으로 채워진 배열 ---")
print(zeros_array)

# 모든 값이 1로 채워진 3행 3열 배열 만들기
ones_array = np.ones((3, 3))
print("\n--- 1로 채워진 배열 ---")
print(ones_array)
```

코딩새싹 🌱: (2, 4) 처럼 튜플로 묶어서 크기를 알려주는군요! 이걸로 미리 공간을 만들어두고 나중에 값을 채워 넣으면 편하겠어요!

선생님: (블록들을 이리저리 바꿔가며) 아주 잘했어! 오늘 우리는 or 논리 마법으로 조건을 유연하게 다루는 법, 넘파이 배열의 모양을 확인하고 자유자재로 변신시키는 .shape과 .reshape(), 그리고 2차원 배열이라는 새로운 세계와 그 안에서 원하는 데이터를 콕콕 집어내거나 잘라내는 방법, 마지막으로 처음부터 특정 값으로 채워진 배열을 만드는 마법까지 배웠어! 이제 넘파이 배열이라는 강력한 도구를 더욱 멋지게 활용할 수 있겠지? 오늘 탐험도 정말 훌륭했어! 🌟