<[파이썬 데이터 마법사] 제1장. 넘파이(NumPy)와 함께 초고속 계산 여행!>

선생님: (커다란 계산기와 숫자 데이터를 상징하는 블록들을 보여주며) 얘들아, 안녕! 우리가 파이썬 리스트로 숫자들을 다뤄봤었지? 예를 들어, 이런 리스트가 있다고 해보자.

```
a = [1, 2, 3, 4]
print("일반 파이썬 리스트 a:", a)
print("a의 타입은?", type(a))
```

선생님: 만약 이 리스트 a의 각 숫자를 제곱해서 [1, 4, 9, 16] 이라는 새 리스트를 만들려면 어떻게 했었지?

코딩새싹 🌱: 앗! for 반복문을 써서, 빈 리스트에 하나씩 제곱한 값을 .append() 했었어요!

```
# a를 활용해서 [1, 4, 9, 16] 제곱 리스트 만들기 (복습!)
b = []
for i in a:
  b.append(i ** 2)
print("for문으로 만든 제곱 리스트 b:", b)
```

선생님: 맞아! 그런데 만약 이 리스트에 숫자가 백만 개, 천만 개가 들어있다면 어떨까? for 반복문이 하나하나 처리하려면 시간이 꽤 걸리겠지? 마치 모래알을 하나씩 세는 것처럼 말이야.

궁금이 💡: 정말 그렇겠네요! 더 빠른 방법은 없나요?

선생님: 바로 그럴 때 등장하는 것이 오늘의 주인공, **넘파이(NumPy)**란다! 넘파이는 “Numerical Python”의 줄임말로, 파이썬에서 대규모 숫자 데이터를 아주 빠르고 효율적으로 계산할 수 있게 도와주는 강력한 라이브러리(도구 모음)야!

<1. 넘파이 소환! 그리고 넘파이 배열 만들기!>

선생님: 넘파이라는 마법 도구를 쓰려면, 먼저 우리 작업 공간으로 불러와야 해. 보통 np라는 별명을 붙여서 부른단다.

```
import numpy as np # 넘파이 라이브러리를 np라는 이름으로 불러와줘!
```

선생님: 이제 우리의 일반 파이썬 리스트 a를 넘파이의 특별한 리스트, 즉 **넘파이 배열(NumPy Array)**로 변신시켜보자!

```
np_list = np.array(a) # 리스트 a를 넘파이 배열로 변신!
```

```
print("\n넘파이 배열 np_list:", np_list)
print("np_list의 타입은?", type(np_list)) # numpy.ndarray 라고 나오지?
```

코딩새싹 🌱: 와! `np.array()`로 감싸주니까 타입이 `<class 'numpy.ndarray'>`로 바뀌었어요! ndarray는 'N차원 배열'이라는 뜻인가요?

선생님: 정확해! 넘파이 배열은 1차원, 2차원, 그 이상의 다차원 숫자 묶음을 아주 효율적으로 다룰 수 있단다.

<2. 넘파이의 슈퍼파워! 한 방에 계산하기 (벡터 연산)!>

선생님: 자, 아까 파이썬 리스트의 각 숫자를 제곱할 때 for문이 필요했지? 넘파이 배열은 얼마나 똑똑한지 한번 볼까?

```
# 넘파이 배열의 모든 숫자를 한 번에 제곱!
squared_np_list = np_list ** 2
print("넘파이 배열 한 방에 제곱:", squared_np_list)
```

궁금이 💡: 헉! for 반복문이 없어졌는데, `np_list ** 2` 하니까 배열 안의 모든 숫자가 다 제곱됐어요! 이게 어떻게 가능한 거예요?

선생님: 그게 바로 넘파이의 슈퍼파워, **벡터 연산(Vectorized Operation)**이란다! 넘파이 배열은 마치 고도로 훈련된 특수부대 같아서, "전체 부대원, 각자 자기 위치에서 제곱 실시!" 라는 명령 하나만 내리면 모든 부대원(배열의 숫자들)이 동시에 자기 임무를 수행하는 거야! for문처럼 한 명씩 불러서 시킬 필요가 없지.

다른 계산도 마찬가지야!

```
my_nums_list = [10, 20, 30, 40, 50]
my_nums_np = np.array(my_nums_list) # 넘파이 배열로 변신!

# 각 숫자에 100을 더하려면?
added_nums_np = my_nums_np + 100
print("각 숫자에 100 더하기:", added_nums_np)
```

<3. 익숙한 친구들: 인덱싱과 슬라이싱>

선생님: 넘파이 배열도 리스트처럼 순서가 있는 데이터 묶음이라서, 우리가 이미 배운 인덱싱과 슬라이싱 기술을 똑같이 쓸 수 있어!

```
print("\n--- 넘파이 배열 인덱싱/슬라이싱 ---")
print("첫 번째 원소:", my_nums_np[0])        # 10
print("0번부터 2번 '전'까지:", my_nums_np[0:2]) # [10 20]
print("거꾸로 출력:", my_nums_np[::-1])    # [50 40 30 20 10]
```

<4. 넘파이의 전용 수학 도구들!>

선생님: 넘파이는 자체적으로 아주 빠르고 편리한 수학 함수들도 많이 가지고 있단다. 예를 들어, 제곱근을 구하는 `np.sqrt()` 같은 게 있지.

```
print("\n--- 넘파이 수학 함수 ---")
print("2의 제곱근:", np.sqrt(2))
# 넘파이 배열 전체에 제곱근을 씌울 수도 있어!
print("my_nums_np 각 원소의 제곱근:", np.sqrt(my_nums_np))
```

코딩새싹 🌱: `np.sqrt()` 하나로 배열 전체에 적용되다니, 정말 편한데요!

선생님: 넘파이 배열끼리의 계산도 아주 간단해!

```
a_np = np.array([1, 2, 3, 4])
b_np = np.array([10, 20, 30, 40])

# 같은 위치의 원소끼리 곱하기
print("원소끼리 곱하기 (방법1):", np.multiply(a_np, b_np))
print("원소끼리 곱하기 (방법2):", a_np * b_np) # 넘파이 배열끼리는 이렇게 간단히!

# 같은 위치의 원소끼리 더하기
print("원소끼리 더하기:", a_np + b_np)
```

<5. 데이터 요약과 탐색: 최댓값, 최솟값, 그리고 `np.where()`!>

선생님: 넘파이는 많은 숫자들 중에서 가장 큰 값이나 작은 값을 찾아내는 것도 아주 잘해!

```
x_list = [1, 7, 10, 3, 10]
x_np = np.array(x_list)
```

```
print("\n--- 데이터 요약 및 탐색 ---")
print("x_np에서 가장 큰 값:", np.max(x_np)) # 또는 x_np.max()
print("x_np에서 가장 작은 값:", np.min(x_np)) # 또는 x_np.min()
```

궁금이 💡: 그럼 특정 조건을 만족하는 숫자가 어디에 있는지 찾아낼 수도 있나요? 예를 들어, x_np 배열에서 값이 10인 애들이 몇 번째 칸에 있는지요!

선생님: 바로 그걸 위해 **np.where()** 라는 탐정 도구가 있단다! 이 탐정은 조건에 맞는 값들의 '위치(인덱스)'를 찾아줘.

```
print("x_np 배열 모습:", x_np) # [ 1  7 10  3 10]


# x_np 배열에서 값이 10인 것들의 위치를 찾아줘!
index_of_tens = np.where(x_np == 10)
print("값이 10인 원소들의 인덱스 정보:", index_of_tens)
```

코딩새싹 🌱: 앗! 결과가 (array([2, 4]),) 이렇게 나왔어요! 튜플 안에 배열이 있고, 그 배열 안에 2랑 4가 있네요! 이게 뭐죠?

선생님: 좋은 관찰이야! np.where()는 결과로 튜플을 돌려주는데, 그 튜플의 첫 번째 항목(index_of_tens[0])이 바로 조건에 맞는 값들의 인덱스들이 담긴 넘파이 배열이란다! 그래서 우리가 찾는 인덱스들은 index_of_tens[0] 안에 들어있지.

- x_np의 2번 인덱스에 10이 있고,
- x_np의 4번 인덱스에 10이 있다는 뜻이야!

만약 두 번째로 10이 나오는 위치를 알고 싶다면?

```
second_ten_index = index_of_tens[0][1] # 첫 번째 배열에서, 두 번째 인덱스(1번) 값을 가져와!
print("두 번째 10의 위치(인덱스):", second_ten_index) # 4가 나오겠지?
print("그 위치의 실제 값:", x_np[second_ten_index])
```

선생님: (블록들을 빠르게 정리하며) 자, 오늘 우리는 넘파이라는 엄청난 데이터 마법사를 만났어! 파이썬 리스트보다 훨씬 빠르고 강력하게 숫자들을 다룰 수 있는 넘파이 배열, 그리고 그 배열로 한 방에 계산하는 벡터 연산까지! 이 넘파이만 있으면 아무리 많은 숫자 데이터라도 두렵지 않겠지? 오늘 정말 새로운 마법을 배우느라 수고 많았어!