<[파이썬 숫자 탐험 고급편] 제2장: 간결한 마법! 람다, 필터, 맵, 그리고 리스트 내포!>

선생님: (작은 마법봉을 휘두르며) 얘들아, 안녕! 우리가 지난 시간에 lambda를 써서 아주 짧은 함수를 만들었지? 두 수를 더하는 함수도 이렇게 한 줄로 뚝딱!

```
# 복습: def로 두 수의 합을 반환하는 함수
def sum_num(a, b):
  return a + b
print(f"def로 만든 함수 결과: {sum_num(10, 3)}")

# lambda 함수로 똑같은 기능!
sums = lambda x, y: x + y # x와 y를 재료로 받아 x + y를 결과로!
print(f"lambda 함수 결과: {sums(2, 6)}")

# 람다 함수는 이름 없이 바로 만들어서 쓸 수도 있어!
# 마치 일회용 마법 주문처럼!
print(f"이름 없는 lambda 결과: {(lambda x, y: x + y)(5, 9)}")
```

코딩새싹 🌱: 와! (lambda x, y: x + y)(5, 9) 이렇게 바로 만들어서 그 자리에서 쓸 수도 있네요! 정말 간편해요!

선생님: 그렇지? 그럼 짝수인지 판별하는 람다 함수도 다시 한번 볼까?

```
is_even = lambda x: x % 2 == 0 # x를 2로 나눈 나머지가 0이면 True!
print(f"8은 짝수인가요? {is_even(8)}")
```

<1. 마법의 체, filter() 함수! - 원하는 것만 골라내기>

선생님: 만약 우리에게 숫자 리스트가 있는데, 그중에서 짝수만 쏙 골라내고 싶다면 어떻게 할까? for문과 if문을 쓸 수도 있지만, filter()라는 마법의 체를 사용하면 더 우아하게 할 수 있단다!

filter(판단_함수, 걸러낼_데이터_묶음)

궁금이 💡: '판단_함수'는 뭐예요? 그 함수가 True를 돌려주는 것만 체에 남는 건가요?

선생님: 정확해! filter는 '판단함수'를 데이터묶음의 각 항목에 적용해서, 결과가 True인 항목들만 남겨주는 마법의 체와 같아. 이때 '판단_함수'는 우리 is_even 람다 함수처럼 True나 False를 돌려주는 함수여야 해.

```
# filter(함수, 리스트 또는 array)
# [1, 2, 3, 6, 8, 7, 11] 리스트에서 짝수만 골라내줘!
even_numbers_iterator = filter(lambda x: x % 2 == 0, [1, 2, 3, 6, 8, 7, 11])
print("filter 결과 (그냥 출력하면?):", even_numbers_iterator) # 어? 이상한 게 나왔네?
print("filter 결과를 리스트로 변신!", list(even_numbers_iterator)) # list()로 감싸야 우리가 아는 리스트가
```

코딩새싹 🌱: filter만 하니까 <filter object at ...> 이렇게 나와요! list()로 한번 더 감싸줘야 리스트가 되는군요!

선생님: 맞아! filter는 일단 "걸러낼 준비가 된 특별한 꾸러미"를 돌려주고, 우리가 list()로 "자, 이제 진짜 리스트로 보여줘!" 해야 한단다.

[퀴즈] num_list = [2, 30, 48, 3, 7, 6, 12] 에서 filter와 lambda를 써서 8 이상인 원소만 뽑아내 보세요!

```
num_list = [2, 30, 48, 3, 7, 6, 12]
result_quiz1 = list(filter(lambda x: x >= 8, num_list))
print("\n8 이상인 숫자들:", result_quiz1)
```

<2. 변신 마법봉, map() 함수! - 모든 항목에 마법 걸기!>

선생님: 그럼 이번엔 리스트의 모든 항목에 똑같은 변신 마법을 걸고 싶다면 어떨까? 예를 들어, 모든 숫자에 10을 곱하고 싶을 때! 이럴 땐 map()이라는 변신 마법봉을 사용한단다.

map(변신_마법_함수, 변신시킬_데이터_묶음)

```
# [1, 2, 5, 6, 9] 리스트의 각 숫자에 10을 곱한 새 리스트를 만들자!
multiplied_iterator = map(lambda x: x * 10, [1, 2, 5, 6, 9])
print("\nmap 결과 (그냥 출력하면?):", multiplied_iterator) # 이것도 이상한 게...
print("map 결과를 리스트로 변신!", list(multiplied_iterator))
```

궁금이 💡: map()도 filter()처럼 바로 리스트가 나오는 게 아니라 특별한 꾸러미를 주는군요! 그럼 map()은 조건으로 거르는 게 아니라, 모든 항목에 함수를 적용해서 '변신'시키는 거네요!

선생님: 정확해! filter는 '선별', map은 '변환'이라고 생각하면 쉽지. 그럼 map을 써서 숫자 리스트를 문자열 리스트로 바꿔볼까? 어떤 함수를 써야 할까?

코딩새싹 🌱: 앗! 숫자를 문자열로 바꾸는 건 str() 함수였죠!

```
nums = [1, 2, 3, 4, 5]
string_iterator = map(str, nums) # str 함수를 각 숫자에 적용!
print("숫자를 문자로 바꾼 결과 (리스트로):", list(string_iterator))
```

[퀴즈] my_list = [1, 2, 3, 4] 의 각 원소를 제곱한 리스트를 map과 lambda를 사용해서 만드시오!

```
my_list = [1, 2, 3, 4]
squared_list = list(map(lambda x: x**2, my_list))
print("\n제곱 리스트 (map 사용):", squared_list)
```

<3. 궁극의 간결함! 리스트 내포(List Comprehension)!>

선생님: lambda, filter, map… 정말 유용한 마법들이지만, 파이썬에는 이 모든 것을 아우르면서 훨씬 더 짧고 읽기 쉽게 표현하는 **리스트 내포**라는 궁극의 마법이 있단다! 마치 긴 주문을 한 단어로 줄여버리는 것과 같지!

기본 형태: [결과표현식 for 항목변수 in 데이터_묶음]
조건 추가: [결과표현식 for 항목변수 in 데이터_묶음 if 조건문]

코딩새싹 🌱: 와! for문이랑 if문이 대괄호 [] 안으로 쏙 들어갔어요!

선생님: 맞아! 한번 볼까? 0부터 9까지 숫자 중 짝수만 골라서 제곱한 리스트를 만들어보자!

```
# 결과표현식: x ** 2
# for 항목변수 in 데이터_묶음: for x in range(10)
# if 조건문: if x % 2 == 0
squared_evens = [x ** 2 for x in range(10) if x % 2 == 0]
print("\n리스트 내포로 만든 짝수 제곱 리스트:", squared_evens)
```

궁금이 💡: 정말 map이랑 filter를 합쳐놓은 것 같아요! "0부터 9까지 x에 대해서, 만약 x가 짝수이면, x의 제곱을 리스트에 넣어줘!" 라고 읽히네요!

선생님: 바로 그거야! 훨씬 직관적이지? 그럼 리스트 내포로 몇 가지 문제를 더 해결해볼까?

[리스트 내포 퀴즈 1] 점수 리스트에서 평균보다 큰 점수들의 (점수 - 평균) 값 구하기!

```
import numpy as np # 평균 계산을 위해 넘파이 소환!
score = [40, 100, 70, 80, 56, 70]
avg = np.average(score) # 평균을 구하고,
print(f"\n평균 점수: {avg:.1f}")

# 각 점수(i)에 대해, 만약 i가 평균(avg)보다 크면, (i - avg)를 반올림해서 리스트에!
deviations = [round(i - avg, 1) for i in score if i > avg]
print("평균 이상 점수들의 편차:", deviations)
```

[리스트 내포 퀴즈 2] 두 리스트 a와 b에서, a에는 있지만 b에는 없는 숫자들만 골라내기! (차집합)

```
a = [1, 2, 4, 5, 6]
b = [3, 5, 9, 10, 6]

# a의 각 숫자(num)에 대해, 만약 num이 b에 없다면, num을 리스트에!
difference_ab = [num for num in a if num not in b]
print("a에만 있는 숫자들 (a - b):", difference_ab)
```

선생님: (마법봉을 내려놓으며) 자, 오늘 우리는 lambda라는 간편 마법, filter라는 선별 마법, map이라는 변환 마법, 그리고 이 모든 것을 아우르는 강력한 리스트 내포 마법까지 배웠어! 이런 마법들을 사용하면 복잡한 데이터 처리도 아주 간결하고 우아하게 해낼 수 있단다. 이제 여러분의 파이썬 코드는 더욱 빛나게 될 거야! 오늘 정말 대단했어! ✨