<[파이썬 수학] 제14장. 마법 주머니 파라미터, *args!>

선생님: (입구가 고무줄처럼 늘어나는 신기한 주머니를 보여주며) 얘들아, 안녕! 지난 시간에 `def sum_nums(a, b):` 처럼 재료(파라미터)를 딱 2개만 받는 함수를 만들었지? 그런데 만약 우리가 숫자 2개를 더할 때도 쓰고, 5개를 더할 때도 쓰고, 100개를 더할 때도 쓸 수 있는 만능 덧셈 함수를 만들고 싶다면 어떻게 해야 할까?

코딩새싹 🌱: 으음… `def sum_nums(a, b, c, d, e...):` 이렇게 파라미터를 엄청 많이 만들어 둬야 하나요?

선생님: 좋은 생각이지만, 몇 개가 필요할지 미리 알 수 없을 땐 곤란하겠지? 그래서 파이썬에는 "몇 개가 들어오든 전부 다 받을게!" 라고 말하는 아주 특별한 파라미터가 있단다. 바로 파라미터 이름 앞에 별표(*)를 붙여주는 거야!

<1. 마법 주머니, *args의 비밀!>

선생님: 보통 이런 '마법 주머니 파라미터'는 `*args` 라고 이름을 짓는 관습이 있어. `args`는 'arguments(인자들)'의 줄임말이지.

```
# *args를 사용하면, 몇 개의 숫자를 넣든 args라는 '튜플'에 전부 담겨!
def sum_nums(*args):
  print("args 주머니에 담긴 재료들:", args) # args가 어떻게 생겼는지 확인해볼까?
  total = 0
  for i in args: # 주머니(튜플)에서 하나씩 꺼내서,
    total += i # total 저금통에 더해준다!
  return total

print("숫자 5개를 더한 결과:", sum_nums(2, 3, 4, 5, 10))
print("숫자 7개를 더한 결과:", sum_nums(2, 3, 4, 5, 10, 23, 100))
```

궁금이 💡: `args`를 출력해보니 소괄호 `()` 로 감싸인 묶음으로 나오네요! `(2, 3, 4, 5, 10)` 이렇게요! 이게 '튜플'인가요?

선생님: 정확해! **튜플(tuple)**은 리스트와 아주 비슷한 순서가 있는 자료형이지만, 한번 만들어지면 내용을 바꿀 수 없다는(immutable) 특징이 있어. 하지만 `for` 반복문으로 하나씩 꺼내 쓰는 건 리스트와 똑같단다! `*` 덕분에 함수를 호출할 때 넣은 모든 인자들이 `args`라는 하나의 튜플 주머니에 쏙 담기는 거지!

<2. *args 응용하기: 여러 리스트의 합집합 & 교집합 구하기!>

선생님: 자, 이 마법 주머니를 이용하면, 지난번에 만든 합집합/교집합 함수를 더 강력하게 업그레이드할 수 있어! 리스트 2개뿐만 아니라, 3개, 4개, 몇 개든 합집합이나 교집합을 구할 수 있게 말이지!

[미션 1] 여러 리스트의 합집합 구하기!

```
# 먼저, 리스트 두 개의 합집합을 구하는 헬퍼(helper) 함수를 만들어두자.
def union(a, b):
  c = a + b
  result = []
  for i in c:
    if i not in result:
      result.append(i)
  return result

# 이제 *args를 이용해서 여러 리스트를 처리하는 메인 함수를 만들자!
def sum_set(*args):
  # args에는 ([1,2,3], [3,4,5], [5,6,7]) 이런 식으로 리스트들이 담겨있겠지?
  u = args[0] # 첫 번째 리스트를 일단 결과물(u)로 정해놓고,
  for i in args[1:]: # 두 번째 리스트부터 하나씩 꺼내서,
    u = union(u, i) # 기존 결과물(u)과 새로운 리스트(i)의 합집합을 구해 다시 u에 저장!
  return u

print("\n세 리스트의 합집합:", sum_set([1, 2, 3, 4], [4, 5, 1, 10], [3, 5, 8, 9]))
```

코딩새싹 🌱: 와! `u = union(u, i)` 이 부분 정말 멋져요! 마치 토너먼트 경기처럼, 첫 번째랑 두 번째가 합쳐진 결과물이 다음 상대랑 또 합쳐지는 거네요!

[미션 2] 여러 리스트의 교집합 구하기!

궁금이 💡: 그럼 교집합도 똑같은 원리로 할 수 있겠네요!

```
# 리스트 두 개의 교집합을 구하는 헬퍼 함수
def inter(a, b):
  result = []
```

```
  for i in a:
    if i in b:
      result.append(i)
  return result

# *args를 이용한 메인 교집합 함수
def inters(*args):
  g = args[0] # 첫 번째 리스트를 기준으로 잡고,
  for i in args[1:]: # 나머지 리스트들과 차례대로 비교하면서
    g = inter(g, i) # 공통된 것들만 남긴다!
  return g

print("세 리스트의 교집합:", inters([1, 4, 5], [2, 3, 4], [3, 4, 5]))
```

〈3. 실전! 함수 활용 퀴즈!〉

선생님: 자, 이제 오늘 배운 것과 이전에 배운 것을 모두 활용해서 문제를 풀어볼 시간이야!

[퀴즈] 두 숫자 1234510과 44393의 자릿수의 합은 얼마일까요?

힌트: 먼저, 숫자를 넣으면 그 숫자의 자릿수를 '숫자'로 돌려주는 함수를 만들어야 해!

코딩새싹 🌱: 으음… 자릿수를 구하는 건 `len(str(숫자))` 였죠! 이걸 함수로 만들면 되겠어요!

```
# 숫자를 넣으면 자릿수(정수)를 돌려주는 함수
def digit_check(n):
  result = len(str(n))
  return result # "7자리 정수" 같은 글자가 아니라, 숫자 7 자체를 돌려줘야 해!

a = 1234510
b = 44393

# digit_check 함수를 두 번 호출해서, 돌려받은 숫자들을 더한다!
total_digits = digit_check(a) + digit_check(b)
```

```
print(f"\n{a}의 자릿수와 {b}의 자릿수의 합은? {total_digits}")
```

궁금이 💡: 아하! digit_check 함수가 만약 f"{result}자리 정수" 처럼 문자열을 return 했다면, “7자리 정수” + “5자리 정수” 가 되어서 계산이 안 됐을 거예요! return 값이 숫자여야 한다는 게 정말 중요하군요!

선생님: (활짝 웃으며) 바로 그거야! 함수의 return 값이 어떤 타입인지 이해하는 것은 정말 중요하단다. 오늘 우리는 *args라는 마법 주머니 파라미터를 사용해서 어떤 개수의 재료든 받을 수 있는 유연한 함수를 만드는 법을 배웠어. 그리고 함수를 만들 때, 그 결과물을 어떻게 재사용할지 생각해서 적절한 타입으로 return해야 한다는 귀중한 통찰력까지 얻었지! 오늘 탐험도 정말 훌륭했어, 우리 꼬마 마법사들! 🌟