<[파이썬 수학] 제13장. 나만의 마법 주문 만들기, 함수(Function)!>

선생님: (두루마리와 깃펜을 들고) 얘들아, 안녕! 우리가 지난 시간까지 구구단을 만들거나, 리스트의 합집합을 구하는 등 멋진 코드들을 만들었지? 그런데 만약 합집합을 구하는 코드를 여기서도 쓰고, 저기서도 써야 한다면 매번 똑같은 코드를 복사해서 붙여넣어야 할까?

코딩새싹 🌱: 으… 너무 번거로울 것 같아요! 실수할 수도 있고요!

선생님: 바로 그거야! 그래서 프로그래머들은 자주 사용하는 코드 묶음에 이름을 붙여서, 필요할 때마다 이름만 불러서 간단히 사용한단다. 이렇게 이름이 붙은 코드 묶음, 즉 '**나만의 마법 주문**'이 바로 **함수(Function)**야!

<1. 마법 주문서 작성법 (함수 정의하기)>

선생님: 함수라는 마법 주문을 만들 때는 정해진 양식이 있어. 마치 요리 레시피를 작성하는 것과 같지.

```
def 마법_주문_이름(재료1, 재료2):
주문_내용(수행할 코드)
return 최종_결과물
```

1. def: "지금부터 새로운 마법 주문을 만들겠습니다!" 라는 선언이야. (define의 약자)
2. **마법주문이름**: 우리가 만든 함수의 이름이지.
3. **(재료1, 재료2)**: 이 주문에 필요한 재료들이야. 이걸 **파라미터(parameter, 매개변수)**라고 불러.
4. **:** 와 **들여쓰기**: 콜론 뒤에 들여쓰기 된 공간이 바로 이 마법 주문의 실제 내용이야.
5. return: 마법을 모두 수행하고 나서, 사용자에게 돌려줄 최종 결과물이지.

```python
# 어떤 숫자를 넣으면 제곱수를 만들어주는 마법 주문
def square_num(num): # 'num'이라는 재료가 하나 필요해!
  return num ** 2 # 재료를 제곱해서 결과물로 돌려줘!


# 마법 주문 사용하기! (함수 호출)
# square_num 이라는 주문에 '5'라는 실제 재료를 넣어 실행!
a = square_num(5) # 함수가 돌려준 결과(25)를 a에 저장!
print("5의 제곱에 10을 뺀 값은?", a - 10)
```

궁금이 💡: 아하! def로 함수를 만드는 건 '레시피를 작성'하는 거고, square_num(5) 처럼 실제로 쓰는 건 '레시피를 보고 요리'하는 거네요! 이때 넣는 5 같은 실제 재료를 **아규먼트(argument, 인자)**라고 부르기도

한단다.

〈2. 가장 중요한 차이! print vs. return〉

선생님: 함수를 만들 때 가장 헷갈리면서도 중요한 것이 바로 print와 return의 차이야.

- **print**: 결과를 화면에 '보여주기만' 하는 확성기. 보여주고 나면 그 결과는 사라져 버려.
- **return**: 결과를 함수 바깥 세상으로 '꺼내서 돌려주는' 택배 상자. 우리는 이 상자를 받아서 다른 계산에 또 쓸 수 있어!

```
def print_sum(a, b):
  print(a + b) # 계산 결과를 확성기로 외치기만 함

def return_sum(a, b):
  return a + b # 계산 결과를 택배 상자에 담아 돌려줌

print("--- print 함수 결과 ---")
result_print = print_sum(10, 21) # 31이라고 외치고, result_print에는 아무것도 담기지 않아.
print("result_print 변수에는 뭐가 들었을까?", result_print) # None (아무것도 없음)

print("\n--- return 함수 결과 ---")
result_return = return_sum(10, 20) # 30이 담긴 택배 상자를 돌려받아 result_return에 저장!
print("result_return 변수에는 뭐가 들었을까?", result_return)
print("결과를 재사용해서 100을 더하면?", result_return + 100)
```

코딩새싹 🌱: 와! print로 만든 함수의 결과는 None이네요! 정말 보여주기만 하고 끝이군요. return으로 만든 함수는 결과를 변수에 저장해서 계속 쓸 수 있고요! 이제 차이를 알겠어요!

〈3. 우리만의 마법 주문 만들기 (실전 퀴즈!)〉

선생님: 자, 이제 우리가 직접 여러 가지 유용한 마법 주문(함수)을 만들어볼 시간이야!

[미션 1] 짝수 판별 마법!

> 숫자를 재료로 받아서, 짝수이면 True, 홀수이면 False를 결과물로 돌려주는 is_even 함수를 만들어보세요!

궁금이 💡: 어떤 수를 2로 나눈 나머지가 0이면 짝수죠! `n % 2 == 0` 이라는 비교 연산 자체가 `True`나 `False`를 돌려주니까, 이걸 바로 `return`하면 되겠어요!

```
def is_even(n):
  return n % 2 == 0 # 이 결과(True 또는 False)를 바로 돌려줘!


print("10은 짝수인가요?", is_even(10))
print("11은 짝수인가요?", is_even(11))
```

[미션 2] 합집합 마법 재사용하기!

지난 시간에 배운 합집합 구하는 코드를, `union_set`이라는 이름의 함수로 만들어보세요!

코딩새싹 🌱: 좋아요! 지난번 코드를 그대로 `def` 안에 넣고, 마지막에 결과 리스트를 `return`하면 되겠어요!

```
def union_set(a, b): # 리스트 a와 b를 재료로 받아서
  c = a + b
  d = []
  for i in c:
    if i not in d:
      d.append(i)
  return d # 중복이 제거된 리스트 d를 결과물로!


x = [1, 2, 3, 4]; y = [3, 4, 5, 6]; z = [1, 6, 10]
print("\nx와 y의 합집합:", union_set(x, y))
print("y와 z의 합집합:", union_set(y, z))
```

궁금이 💡: 와! 함수로 만들어두니까 `union_set()` 이름만 부르면 어떤 리스트든 합집합을 바로 구할 수 있네요! 정말 편해요!

[미션 3] 구구단 출력 마법!

숫자를 하나 재료로 받으면, 그 숫자에 해당하는 구구단을 출력하는 `multi_table` 함수를 만들고,

이 함수를 이용해서 2단부터 9단까지 모두 출력해보세요!

선생님: `for` 반복문 안에서 우리가 만든 함수를 호출하면, 반복 작업이 훨씬 간단해지겠지?

```
def multi_table(n):
  print(f"\n--- {n}단 ---")
  for i in range(1, 10):
    print(f"{n} * {i} = {n * i}")


# 2부터 9까지의 숫자를 하나씩 꺼내서 multi_table 함수의 재료로 사용!
for i in range(2, 10):
  multi_table(i)
```

코딩새싹 🌱: for문으로 multi_table()을 8번 호출하니까 구구단 전체가 뚝딱 만들어졌어요! 정말 대단해요!

<마법 주문의 주의사항!>

선생님: 함수라는 마법 주문을 쓸 때는 약속을 잘 지켜야 해. 레시피에 재료가 2개 필요하다고 적혀있는데 1개만 넣으면 요리를 할 수 없겠지?

```
def sum_nums(a, b):
  return a + b
# c = sum_nums(4) # 재료(파라미터)가 2개 필요한데, 1개만 넣어서 TypeError 발생!
```

궁금이 💡: 아! a와 b 두 개가 필요한데 4 하나만 넣어서 에러가 나는군요! 그럼 함수는 자기가 모르는 재료를 쓸 수도 없겠네요?

선생님: 정확해! 함수는 오직 **자신이 전달받은 재료(파라미터)와 함수 안에서 직접 만든 변수**만 알고 있단다. 이걸 **스코프(Scope, 유효 범위)**라고 해.

```
c = 100 # 바깥 세상에 있는 c
def sum_of_two(a, b):
  return a + b + c # 함수는 자기가 전달받지 않은 바깥 세상의 c를 모름!
# sum_of_two(20, 35) # 실행하면 c를 모른다고 NameError 발생!
```

선생님: (두루마리를 말며) 자, 오늘 우리는 코드를 재사용하고, 프로그램을 더 깔끔하게 만들어주는 '함수'라는 위대한 마법을 배웠어! def로 정의하고, return으로 결과를 돌려받는 원리, 그리고 print와의 차이점까지! 이제 여러분은 자신만의 마법 주문을 만들 수 있는 진짜 마법사가 되었단다! 오늘 정말 수고 많았어! 🌟