<제10장. 함수를 부르는 마법 상자, 계산기를 만들자!>

선생님: (손뼉을 치며) 코딩새싹 🌱, 궁금이 💡! 우리가 지금까지 함수도 배우고, 딕셔너리도 배웠지? 오늘은 이 두 가지를 합쳐서 아주 특별한 마법을 부려볼 거야! 바로, 우리가 직접 숫자를 입력하면 계산 결과를 알려주는 '나만의 계산기'를 만드는 것이지!

코딩새싹 🌱: 계산기요? 우와! 더하기, 빼기, 곱하기, 나누기를 다 할 수 있는 건가요? 😮

선생님: 물론이지! 먼저, 계산기의 핵심 부품이 될 더하기, 빼기, 곱하기, 나누기 기능을 하는 함수들을 만들어야겠지? 이건 우리가 예전에 배운 함수의 모습 그대로야.

```
def add(a, b):
  return a + b


def subtract(a, b): # substract -> subtract 로 수정했어요!
  return a - b


def multiply(a, b):
  return a * b


def divide(a, b):
  return a / b
```

궁금이 💡: 아, 네! add 함수는 두 수를 받아서 더한 값을 돌려주고, subtract 함수는 뺀 값을 돌려주고… 익숙한 모습이에요!

선생님: 아주 좋아! 자, 이제 마법 상자를 만들 차례야. 이 마법 상자는 사용자가 "+" 기호를 보여주면 add 함수를, "-" 기호를 보여주면 subtract 함수를 뽕! 하고 꺼내주는 역할을 할 거란다. 이 마법 상자가 바로… 딕셔너리야!

```
operations = {
    "+": add,
    "-": subtract,
    "*": multiply,
    "/": divide
```

}

코딩새싹 🌱: 네? 딕셔너리에 함수가 들어갔어요? 키는 "+" 같은 문자열인데, 값으로 add 같은 함수 이름이 쓰여있네요! 신기해요! 😲

선생님: (미소 지으며) 바로 그게 핵심이야! 파이썬에서는 함수도 값처럼 취급될 수 있어서, 이렇게 딕셔너리의 값으로 저장해 둘 수 있단다. add 라고 쓰면 add() 처럼 함수를 실행하라는 게 아니라, add 라는 함수 그 자체를 가리키는 거야. 마치 함수의 이름표를 딕셔너리에 붙여둔 것과 같지.

궁금이 💡: 그럼 operations["+"] 라고 하면 add 함수가 나오는 건가요? 그럼 그 함수를 바로 실행할 수도 있어요?

선생님: 정확해! (주석 처리된 코드를 가리키며) 여기 봐봐. 만약 우리가 이렇게 한다면…

```
# function = operations["-"] # operations 딕셔너리에서 "-" 키로 subtract 함수를 꺼내 function 변수에 저장
# result = function(5, 2)    # function은 이제 subtract 함수니까, 5 - 2를 실행!
# print(result)              # 결과로 3이 나오겠지?
```

코딩새싹 🌱: 와! operations 딕셔너리가 마치 함수 자판기 같아요! 기호 버튼을 누르면 해당 기능의 함수가 툭 튀어나오는 거예요!

<1단계: 한 번 계산하는 계산기 만들기!>

선생님: 자, 그럼 이 함수 자판기를 이용해서 첫 번째 계산기를 만들어볼까? 먼저 사용자에게서 첫 번째 숫자와 연산 기호, 그리고 두 번째 숫자를 입력받아야겠지?

```
# (첫 번째 계산기 코드)
num1 = int(input("첫번째 수를 입력하세요: ")) # 1. 첫 번째 숫자 입력

print("사용 가능한 연산 기호:")
for symbol in operations: # 2. operations 딕셔너리의 키(연산 기호)들을 보여주기
  print(symbol)

operation_symbol = input("위 연산기호 중 하나를 입력하세요: ") # 3. 사용할 연산 기호 입력
num2 = int(input("다음 수를 입력하세요: ")) # 4. 두 번째 숫자 입력
```

```
# 5. 마법의 순간!
my_function = operations[operation_symbol] # 입력받은 기호로 딕셔너리에서 함수를 꺼내요!
result = my_function(num1, num2) # 꺼낸 함수에 두 숫자를 넣어 실행!

print(f"{num1} {operation_symbol} {num2} = {result}") # 6. 결과 출력!
```

궁금이 💡: `my_function = operations[operation_symbol]` 이 부분에서, 만약 제가 "+"를 입력했으면 `my_function`은 add 함수가 되는 거군요! 그리고 바로 밑에서 `my_function(num1, num2)` 이렇게 호출하면 `add(num1, num2)`가 실행되는 거고요!

선생님: 완벽해! 바로 그렇게 동작하는 거란다. `input()`으로 받은 숫자는 문자열이니까 `int()`로 정수형으로 바꿔주는 것도 잊지 않았지. 자, 이렇게 하면 한 번 계산하고 프로그램이 끝나.

<2단계: 계속 계속 계산하는 업그레이드 계산기!>

선생님: 그런데 계산기는 보통 한 번만 쓰고 끝나지 않지? 여러 번 계속 계산하고 싶을 때도 있잖아. 그러려면 어떻게 해야 할까?

코딩새싹 🌱: 음… 반복문! `while` 반복문을 쓰면 되지 않을까요?

선생님: 정답! `while` 반복문으로 계산 과정을 감싸고, 사용자가 그만하고 싶을 때 빠져나올 수 있도록 만들어주면 돼. 그리고 더 멋진 건, 이전 계산 결과를 다음 계산의 첫 번째 숫자로 바로 사용할 수 있게 하는 거야!

```
# (두 번째, 업그레이드된 계산기 코드)
def add(a, b): return a + b
def subtract(a, b): return a - b # 아까 정의했지만, 전체 코드 흐름상 다시 보여줄게!
def multiply(a, b): return a * b
def divide(a, b): return a / b

operations = {
    "+": add, "-": subtract, "*": multiply, "/": divide
}

num1 = int(input("첫번째 수를 입력하세요: ")) # 맨 처음 첫 번째 숫자는 한 번만 입력받아요.
```

```
running = True # 계산기를 계속 실행할지 정하는 스위치!
while running: # 스위치가 True인 동안 계속 반복!
  print("\n사용 가능한 연산 기호:")
  for symbol in operations:
    print(symbol)

  operation_symbol = input("연산기호 중 하나를 입력하세요(+-*/), 계산을 마치려면 'n'을 입력하세요: ")

  if operation_symbol == "n": # 만약 'n'을 입력하면,
    running = False # 스위치를 False로 바꿔서 반복문을 멈춰요!
    print("계산기를 종료합니다.")
  elif operation_symbol in operations: # 입력한 기호가 우리 딕셔너리에 있는 유효한 기호라면,
    num2 = int(input("다음 수를 입력하세요: "))

    my_function = operations[operation_symbol]
    result = my_function(num1, num2)
    print(f"{num1} {operation_symbol} {num2} = {result}")

    num1 = result # ★ 여기가 중요! 다음 계산을 위해 첫 번째 숫자를 현재 결과로 업데이트! ★
  else:
    print("잘못된 연산 기호입니다. 다시 시도해주세요.")
```

궁금이 💡: 와! `while running:` 루프가 생겼네요! 그리고 `operation_symbol == "n"`이면 `running = False`가 돼서 루프를 빠져나오는 거군요!

코딩새싹 🌱: 선생님! 저기 별표(★) 쳐진 `num1 = result` 이 부분은 뭐예요?

선생님: 아주 중요한 부분을 발견했구나! 저 코드가 바로 계산기를 더욱 계산기답게 만들어주는 마법이란다. 만약 우리가 `5 + 3`을 계산해서 8이라는 결과를 얻었다고 해보자. 그 다음 바로 `8 * 2`를 계산하고 싶다면, 다음 계산의 첫 번째 숫자는 방금 얻은 결과인 8이 되어야겠지? `num1 = result`는 바로 그 역할을 하는 거야. 이번 계산의 결과를 다음 계산의 첫 번째 숫자로 만들어주는 거지!

궁금이 💡: 아하! 그래서 루프가 다시 돌 때 num1은 이전 결과값을 가지고 시작하는 거군요! 예를 들어 10을 입력하고 +를 누르고 5를 입력하면 결과가 15가 나오고, num1이 15가 되는 거죠? 그 다음에 *를 누르고 2를 입력하면 15 * 2 = 30 이렇게 계산되는 거고요!

선생님: (흐뭇하게 웃으며) 바로 그거란다! 이제 여러분은 함수를 딕셔너리에 저장해서 필요에 따라 꺼내 쓰고, 반복문과 조건문을 활용해서 사용자와 계속 소통하며, 이전 결과를 다음 계산에 활용하는 멋진 계산기 프로그램을 만들 수 있게 되었어! 어때, 정말 신기하고 재미있지 않니?

코딩새싹 🌱: 네, 선생님! 함수가 값처럼 쓰일 수 있다는 것도 신기하고, 그걸로 계산기를 만드니 정말 프로그래머가 된 것 같아요!

궁금이 💡: 저도요! 이제 여기에 더 많은 기능을 추가해보고 싶어요! 예를 들면 제곱 기능이나 나머지 구하는 기능 같은 거요! operations 딕셔너리에 새로운 기호랑 함수만 추가하면 되겠죠?

선생님: (박수를 치며) 아주 훌륭한 생각이야! 그렇게 스스로 기능을 확장해 나가는 것이 바로 코딩의 즐거움이지! 오늘 여러분은 정말 큰 걸음을 내디뎠단다. 대단해!