파이썬의 만능 해결사, 함수! 🤖✨📜

등장인물:

- **슬기 선생님:** 친절하고 유머러스한 코딩 선생님
- **지혜:** 똑똑하고 질문이 많은 학생
- **코딩이:** 코딩을 처음 배우지만 호기심 많은 학생

슬기 선생님: 여러분, 안녕하세요! 우리가 지금까지 `print()`, `range()`, `Turtle()`, `forward()`, `circle()` 등등 정말 많은 명령어들을 사용해 봤죠? 이런 것들이 사실은 파이썬에 미리 만들어져 있는 작은 프로그램, 즉 **함수(Function)**랍니다! 오늘은 우리가 직접 이런 편리한 함수를 만들어보는 방법을 배울 거예요!

코딩이: 함수요? 그게 뭐예요? `Turtle()`이나 `print()`처럼 괄호가 붙는 애들이 다 함수예요? 우리가 직접 만들 수도 있다고요? 우와! 그럼 저도 코딩이만세() 이런 함수 만들 수 있어요?

슬기 선생님: (웃음) 네, 코딩이 학생! 물론 코딩이만세() 함수도 만들 수 있어요! 함수는 아주 간단하게 말하면, **어떤 특별한 기능을 하는 코드들의 묶음**이에요. 마치 우리가 요리할 때 사용하는 '요리법(레시피)' 같다고 생각하면 쉬워요. 특정 요리를 만들기 위한 재료와 순서가 적혀있는 것처럼, 함수도 어떤 기능을 수행하기 위한 코드들이 순서대로 적혀있는 거죠.

지혜: 요리법이요? 그럼 함수를 만들어두면 똑같은 요리(기능)를 하고 싶을 때마다 그 요리법(함수)만 가져다 쓰면 되니까 편하겠네요!

슬기 선생님: 맞아요, 지혜 학생! 그게 바로 함수를 쓰는 가장 큰 이유 중 하나예요. 똑같은 코드를 여러 번 써야 할 때, 그걸 함수로 딱 한 번만 만들어두면 필요할 때마다 그 함수의 이름만 불러서 간단하게 사용할 수 있거든요. 자, 그럼 함수라는 요리법은 어떻게 만드는지 한번 볼까요?

1. 함수 정의하기: 나만의 요리법 만들기!

슬기 선생님: 함수를 만들 때는 "자, 지금부터 새로운 요리법을 만들 거야!" 하고 파이썬에게 알려줘야 해요. 이때 사용하는 특별한 약속이 바로 `def`랍니다.

```
def add(a, b):        # 'add'라는 이름의 함수를 만들 거야! 이 함수는 'a'와 'b'라는 두 가지 정보가 필요해.
  result = a + b    # 'a'랑 'b'를 더해서 'result'에 저장하고,
  return result     # 그 'result' 값을 알려줄게! (이것이 이 함수의 결과물이야)
```

지혜: `def add(a, b):` 이렇게 쓰는군요! `def`가 "정의한다(Define)"는 뜻인가 봐요. `add`는 함수 이름이고, 괄호 안에 있는 `a`랑 `b`는 뭐예요? 그리고 다음 줄부터는 들여쓰기를 해야 하네요!

슬기 선생님: 아주 잘 봤어요!

- `def`는 "지금부터 함수를 만들겠습니다!" 라는 신호예요.
- `add`는 우리가 만든 함수의 이름이죠. 우리가 직접 지어줄 수 있어요.
- 괄호 `()` 안에 있는 `a`와 `b`는 이 `add` 함수가 일을 하기 위해 **필요한 정보(재료)**들이에요. 이걸 어려운 말로 **파라미터(Parameter)** 또는 **매개변수**라고 하는데, 그냥 "함수가 받아야 할 빈칸" 또는 "정보 쪽지"라고 생각해도 좋아요. `add` 함수는 두 개의 정보 쪽지(`a`와 `b`)를 받아야 일을 할 수 있는 거죠.

- 그리고 콜론(:)을 꼭 찍어주고, 다음 줄부터는 이 add 함수가 할 일들을 **들여쓰기** 해서 순서대로 적어주는 거예요. 이 들여쓰기 된 부분이 바로 add 함수의 실제 요리법 내용(코드 블록)이랍니다.
- 마지막에 있는 `return result`는 이 add 함수가 모든 일을 마치고 나서 "자, 이게 내가 만든 결과물이야!" 하고 알려주는 거예요. `return` 뒤에 있는 값을 함수의 최종 결과로 돌려주는 거죠. 모든 함수가 꼭 `return`을 해야 하는 건 아니지만, 계산 결과를 알려줘야 할 때는 필요하답니다.

코딩이: 아하! `def add(a, b):`는 "두 개 더하기 요리법"을 만드는 거고, a랑 b는 그 요리에 필요한 재료 두 가지네요! 그럼 저 요리법을 컴퓨터가 언제 만들어요? 코드를 위에서부터 쭉 읽으면서 바로 만들어요?

슬기 선생님: 아주 중요한 질문이에요, 코딩이 학생! 파이썬이 `def add(a, b): ...` 이 부분을 처음 읽을 때는, 요리법을 실제로 요리하지 않아요. 그냥 **"아, add라는 이름의 요리법이 있고, 그건 두 가지 재료(a, b)가 필요하고, 이런이런 순서로 요리해서 결과를 알려주는구나!" 하고 요리법 자체를 배우고 기억만 해둬요.** 마치 우리가 요리책을 읽고 '이런 레시피가 있군!' 하고 알아두는 것과 같아요. 아직 요리를 시작한 건 아니죠?

2. 함수 호출하기: 요리법대로 실제로 요리하기!

슬기 선생님: 이렇게 만들어둔(정의해둔) 함수라는 요리법은, 우리가 "자, 이제 이 요리법대로 요리해줘!" 하고 **불러줄(호출할) 때** 비로소 실제로 일을 시작해요.

```
sum_value = add(2, 4) # 'add' 요리법(함수)을 사용해서 2랑 4를 가지고 요리해줘!
print(sum_value)      # 요리 결과(sum_value)를 화면에 보여줘!
```

지혜: `add(2, 4)` 이렇게 함수의 이름이랑 괄호 안에 실제 숫자 2랑 4를 넣어서 불렀네요! 이걸 **함수 호출(Call)**이라고 하는군요! 그럼 아까 add 함수 만들 때 있던 a랑 b에는 뭐가 들어가요?

슬기 선생님: 정확해요! `add(2, 4)` 이렇게 함수를 부르면,

1. 파이썬은 기억해뒀던 add 요리법을 찾아가요.
2. 우리가 전달한 **실제 재료(값)**인 2를 add 요리법의 첫 번째 정보 쪽지(파라미터) a에 쏙 넣어줘요. 그래서 a는 2가 돼요.
3. 두 번째 실제 재료인 4는 두 번째 정보 쪽지 b에 쏙! 그래서 b는 4가 되죠.
4. 이제 add 함수 안의 요리법대로 `result = a + b` (즉, `result = 2 + 4`)가 실행되고, `result`는 6이 돼요.
5. 그리고 `return result`를 만나서 6이라는 결과물을 `add(2, 4)`를 불렀던 자리로 돌려줘요.
6. 그래서 `sum_value = add(2, 4)` 코드에서 `add(2, 4)` 부분이 6으로 바뀌니까, `sum_value`라는 변수에는 6이 저장되는 거랍니다!

코딩이: 와! 함수를 정의할 때는 그냥 "이런 기능이 있어~" 하고 알려만 주는 거고, 실제로 부를 때(add(2,4) 할 때) 그 안에 있는 코드들이 움직이는 거네요! 신기하다! 그럼 재료(파라미터) 개수랑 실제로 주는 값(인자) 개수는 똑같아야 해요? add 함수는 재료가 a, b 두 개인데, `add(2)` 이렇게 하나만 주면 어떻게 돼요?

슬기 선생님: 코딩이 학생, 정말 중요한 점을 짚었어요! 네, 보통은 함수를 만들 때 정해둔 **필요한 정보(파라미터)의 개수**와 함수를 부를 때 주는 **실제 정보(인자, Argument)의 개수**가 같아야 해요. add 함수는 a와 b 두 개의 정보를 받기로 약속했는데, 만약 `add(2)`처럼 하나만 주면 "어? 나머지 한 개 재료는 어디 갔지? 요리를 못 하겠어!" 하면서 오류가 날 수 있답니다.

3. 다양한 함수 예시들

슬기 선생님: 자, 그럼 다른 예시들도 살펴볼까요?

- **어떤 수를 입력받아서 10배를 돌려주는 함수**

```
def multiple_ten(number_one): # 'number_one' 이라는 정보 하나가 필요해요.
  return number_one * 10    # 받은 정보에 10을 곱해서 돌려줄게요.

result_ten = multiple_ten(2) # multiple_ten 함수에 2를 넣어서 호출!
print(result_ten)            # 결과는 20이 나오겠죠?
```

지혜: multiple_ten 함수는 number_one이라는 이름으로 정보 하나를 받네요! multiple_ten(2)라고 부르면 number_one에 2가 들어가서 2 곱하기 10을 한 20을 돌려주고요!

- **정다각형 만드는 함수 (우리 거북이 출동!)**

```
# (이전에 clearscreen(), king = Turtle() 등등 설정은 되어 있다고 가정할게요!)
# def polygon(sides_count): # 'sides_count'라는 정보(몇 각형인지)가 필요해요.
#   for i in range(sides_count):
#     king.forward(80)
#     king.right(360 / sides_count) # 받은 정보대로 각도 계산!

# polygon(8) # 8각형을 그려줘! 라고 함수 호출!
# polygon(5) # 이번엔 5각형을 그려줘!
```

(실제 거북이 코드는 다음과 같이 완전하게 제시)

```
clearscreen()

king = Turtle()
king.speed(10)
king.pensize(5)
king.color("green")

def polygon(sides_count): # 'sides_count'는 몇 각형을 그릴지 알려주는 정보예요.
  angle = 360 / sides_count # 정다각형의 외각 크기를 계산해요.
  for i in range(sides_count): # 'sides_count' 만큼 반복하면서 변을 그려요.
    king.forward(80)
    king.right(angle)

polygon(8) # 자, 이제 'polygon' 함수를 불러서 8각형을 그려볼까요?
           # 'sides_count'에 8이 전달돼요!
king.penup()
king.goto(150,0) # 위치를 옮겨서
king.pendown()
```

```
polygon(5) # 이번엔 'polygon' 함수를 불러서 5각형을 그려요!
           # 'sides_count'에 5가 전달돼요!
```

코딩이: 와! `polygon` 함수는 `sides_count`라는 정보(몇 각형 그릴지)를 받아서, 그 숫자만큼 반복하면서 앞으로 가고 꺾고 하네요! `polygon(8)`이라고 부르면 8각형이 그려지고, `polygon(5)`라고 부르면 5각형이 그려져요! 똑같은 그림 그리는 코드를 여러 번 안 써도 되니까 진짜 편한데요?

슬기 선생님: 그렇죠? `polygon` 함수는 `return`으로 값을 돌려주지는 않지만, 대신 거북이를 움직여서 그림을 그리는 멋진 **기능**을 수행했어요. 이렇게 함수는 반복되는 코드를 한 곳에 모아두고, 필요할 때마다 이름만 불러서 쉽게 사용할 수 있게 해주는 아주 강력한 도구랍니다! "정의"할 때는 레시피를 만들기만 하고, "호출"할 때 비로소 그 레시피대로 행동한다는 것, 그리고 필요한 정보를 정확히 전달해야 한다는 것만 잘 기억하면 여러분도 멋진 함수들을 많이 만들 수 있을 거예요!

지혜 & 코딩이: 네! 선생님! 이제 함수가 뭔지 확실히 알겠어요! 저희도 직접 유용한 함수를 만들어보고 싶어요!