재귀함수 기초 연습 - Easy

기초 재귀 연습 (Apprentice)

1. 재귀함수를 사용하여 리스트(배열)의 각 원소를 한 줄씩 출력하는 함수를 작성하세요. 예: [1, 2, 3]을 입력하면 1, 2, 3이 각각 한 줄씩 출력

```
def print_array(arr):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print_array([1, 2, 3, 4, 5])
```

입력예시

```
[1, 2, 3, 4, 5]
```

출력예시

```
1
2
3
4
5
```

> **힌트:** *기저 조건: 리스트가 비어있으면 종료합니다. 첫 번째 원소를 출력한 다음, print_array(arr[1:])을 호출합니다.*

해설

```
def print_array(arr):
    # 기저 조건: 빈 리스트이면 종료
    if not arr:
        return
    # 첫 번째 원소 출력
    print(arr[0])
    # 재귀 호출: 나머지 원소들 출력
    print_array(arr[1:])

# 동작 과정:
# print_array([1,2,3,4,5]): print(1) → print_array([2,3,4,5])
# print_array([2,3,4,5]): print(2) → print_array([3,4,5])
# print_array([3,4,5]): print(3) → print_array([4,5])
# print_array([4,5]): print(4) → print_array([5])
# print_array([5]): print(5) → print_array([])
# print_array([]): 종료
```

2. 재귀함수를 사용하여 1부터 n까지 카운트업하며 출력하는 함수를 작성하세요. 예: n=5일 때 1, 2, 3, 4, 5를 각각 한 줄씩 출력

```
def countup(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
countup(5)
```

입력예시

```
5
```

출력예시

```
1
2
3
4
5
```

> **힌트:** *기저 조건: n이 0 이하이면 종료합니다. 먼저 countup(n-1)을 호출한 다음, n을 출력합니다.*

해설

```
def countup(n):
    # 기저 조건: n이 0 이하이면 종료
    if n <= 0:
        return
    # 재귀 호출: 먼저 1부터 n-1까지 출력
    countup(n - 1)
    # 현재 숫자를 출력
    print(n)

# 동작 과정:
# countup(5) → countup(4) → ... → countup(1) → countup(0)
# countup(0): 종료
# countup(1): print(1)
# countup(2): print(2)
# countup(3): print(3)
# countup(4): print(4)
# countup(5): print(5)
```

3. 재귀함수를 사용하여 n!(팩토리얼)을 계산하는 함수를 작성하세요. 팩토리얼은 n! = n × (n-1) × (n-2) × ... × 1 입니다. 예: 5! = 5 × 4 × 3 × 2 × 1 = 120

```
def factorial(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(factorial(5))  # 120이 나와야 합니다
```

입력예시

```
5
```

출력예시

```
120
```

> **힌트:** *기저 조건: n이 0 또는 1일 때는 1을 반환합니다. 재귀 호출: n * factorial(n-1)을 반환합니다.*

해설

```
def factorial(n):
    # 기저 조건: n이 0 또는 1이면 1 반환
    if n <= 1:
        return 1
    # 재귀 호출: n * (n-1)!
    return n * factorial(n - 1)

# 동작 과정 예시:
# factorial(5) = 5 * factorial(4)
#              = 5 * 4 * factorial(3)
#              = 5 * 4 * 3 * factorial(2)
#              = 5 * 4 * 3 * 2 * factorial(1)
#              = 5 * 4 * 3 * 2 * 1
#              = 120
```

4. 재귀함수를 사용하여 주어진 양의 정수의 각 자릿수를 모두 곱한 값을 계산하는 함수를 작성하세요. 예: 1234의 경우 1×2×3×4 = 24

```
def product_of_digits(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(product_of_digits(1234))  # 24가 나와야 합니다
print(product_of_digits(567))   # 210이 나와야 합니다
```

입력예시

```
1234
```

출력예시

```
24
```

> **힌트:** *기저 조건: n이 0이면 1을 반환합니다. 재귀 호출: (n%10) * product_of_digits(n//10)을 반환합니다.*

해설

```
def product_of_digits(n):
    # 기저 조건: n이 0이면 1 반환 (곱셈의 항등원)
    if n == 0:
        return 1
    # 재귀 호출: 마지막 자릿수 × 나머지 자릿수들의 곱
    return (n % 10) * product_of_digits(n // 10)

# 동작 과정 예시:
# product_of_digits(1234) = 4 × product_of_digits(123)
#                         = 4 × 3 × product_of_digits(12)
#                         = 4 × 3 × 2 × product_of_digits(1)
#                         = 4 × 3 × 2 × 1 × product_of_digits(0)
#                         = 4 × 3 × 2 × 1 × 1
#                         = 24
```

5. 재귀함수를 사용하여 1^2부터 n^2까지의 합을 계산하는 함수를 작성하세요. 제곱수의 합은 $1^2 + 2^2 + 3^2 + \ldots + n^2$ 입니다. 예: n=4일 때 $1^2+2^2+3^2+4^2 = 1+4+9+16 = 30$

```
def sum_of_squares(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(sum_of_squares(4))  # 30이 나와야 합니다
print(sum_of_squares(3))  # 14가 나와야 합니다
```

입력예시

```
4
```

출력예시

```
30
```

> **힌트:** *기저 조건: n이 0 이하이면 0을 반환합니다. 재귀 호출: n*n + sum_of_squares(n-1)을 반환합니다.*

해설

```
def sum_of_squares(n):
    # 기저 조건: n이 0 이하이면 0 반환
    if n <= 0:
        return 0
    # 재귀 호출: n² + (1²부터 (n-1)²까지의 합)
    return n * n + sum_of_squares(n - 1)

# 동작 과정 예시:
# sum_of_squares(4) = 4² + sum_of_squares(3)
#                   = 16 + 3² + sum_of_squares(2)
#                   = 16 + 9 + 2² + sum_of_squares(1)
#                   = 16 + 9 + 4 + 1² + sum_of_squares(0)
#                   = 16 + 9 + 4 + 1 + 0
#                   = 30
```

6. 재귀함수를 사용하여 별(*)로 삼각형을 출력하는 함수를 작성하세요. n줄짜리 삼각형에서 i번째 줄에는 i개의 별을 출력합니다.

```
def print_stars(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print_stars(4)
```

입력예시

```
4
```

출력예시

```
*
**
***
****
```

> **힌트:** *기저 조건: n이 0 이하이면 종료합니다. 먼저 print_stars(n-1)을 호출한 다음, '*' * n을 출력합니다.*

해설

```
def print_stars(n):
    # 기저 조건: n이 0 이하이면 종료
    if n <= 0:
        return
    # 재귀 호출: 먼저 n-1줄까지 출력
    print_stars(n - 1)
    # 현재 줄에 n개의 별 출력
    print('*' * n)

# 동작 과정:
# print_stars(4) → print_stars(3) → ... → print_stars(0)
# print_stars(0): 종료
# print_stars(1): print('*')
# print_stars(2): print('**')
# print_stars(3): print('***')
# print_stars(4): print('****')
```

7. 재귀함수를 사용하여 주어진 양의 정수의 자릿수 개수를 계산하는 함수를 작성하세요. 예: 1234는 4자리, 56은 2자리

```
def count_digits(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(count_digits(1234))  # 4가 나와야 합니다
print(count_digits(56))    # 2가 나와야 합니다
print(count_digits(7))     # 1이 나와야 합니다
```

입력예시

```
1234
```

출력예시

```
4
```

> **힌트:** *기저 조건: n이 한 자릿수(n < 10)이면 1을 반환합니다. 재귀 호출: 1 + count_digits(n//10)을 반환합니다.*

해설

```
def count_digits(n):
    # 기저 조건: 한 자릿수이면 1 반환
    if n < 10:
        return 1
    # 재귀 호출: 1 + (마지막 자릿수를 제거한 숫자의 자릿수)
    return 1 + count_digits(n // 10)

# 동작 과정 예시:
# count_digits(1234) = 1 + count_digits(123)
#                    = 1 + 1 + count_digits(12)
#                    = 1 + 1 + 1 + count_digits(1)
#                    = 1 + 1 + 1 + 1
#                    = 4
```

8. 재귀함수를 사용하여 리스트(배열)의 모든 원소의 곱을 계산하는 함수를 작성하세요. 예: [2, 3, 4]의 곱은 24

```
def array_product(arr):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(array_product([2, 3, 4]))    # 24가 나와야 합니다
print(array_product([1, 5, 2]))    # 10이 나와야 합니다
print(array_product([]))           # 1이 나와야 합니다
```

입력예시

```
[2, 3, 4]
```

출력예시

```
24
```

> **힌트:** *기저 조건: 리스트가 비어있으면 1을 반환합니다(곱셈의 항등원). 재귀 호출: arr[0] * array_product(arr[1:])을 반환합니다.*

해설

```
def array_product(arr):
    # 기저 조건: 빈 리스트이면 1 반환 (곱셈의 항등원)
    if not arr:
        return 1
    # 재귀 호출: 첫 번째 원소 × 나머지 원소들의 곱
    return arr[0] * array_product(arr[1:])

# 동작 과정 예시:
# array_product([2,3,4]) = 2 × array_product([3,4])
#                        = 2 × 3 × array_product([4])
#                        = 2 × 3 × 4 × array_product([])
#                        = 2 × 3 × 4 × 1
#                        = 24
```

9. 재귀함수를 사용하여 리스트(배열)의 원소 개수를 계산하는 함수를 작성하세요. (len() 함수를 사용하지 않고) 예: [1, 2, 3, 4, 5]의 원소 개수는 5

```
def array_length(arr):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(array_length([1, 2, 3, 4, 5]))  # 5가 나와야 합니다
print(array_length([10, 20]))         # 2가 나와야 합니다
print(array_length([]))               # 0이 나와야 합니다
```

입력예시

```
[1, 2, 3, 4, 5]
```

출력예시

```
5
```

> **힌트:** *기저 조건: 리스트가 비어있으면 0을 반환합니다. 재귀 호출: 1 + array_length(arr[1:])을 반환합니다.*

해설

```
def array_length(arr):
    # 기저 조건: 빈 리스트이면 0 반환
    if not arr:
        return 0
    # 재귀 호출: 1 + (첫 번째 원소를 제거한 리스트의 길이)
    return 1 + array_length(arr[1:])

# 동작 과정 예시:
# array_length([1,2,3,4,5]) = 1 + array_length([2,3,4,5])
#                           = 1 + 1 + array_length([3,4,5])
#                           = 1 + 1 + 1 + array_length([4,5])
#                           = 1 + 1 + 1 + 1 + array_length([5])
#                           = 1 + 1 + 1 + 1 + 1 + array_length([])
#                           = 1 + 1 + 1 + 1 + 1 + 0
#                           = 5
```

10. 재귀함수를 사용하여 n부터 1까지 카운트다운하며 출력하는 함수를 작성하세요. 예: n=5일 때 5, 4, 3, 2, 1을 각각 한 줄씩 출력

```
def countdown(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
countdown(5)
```

입력예시

```
5
```

출력예시

```
5
4
3
2
1
```

> **힌트:** *기저 조건: n이 0 이하이면 종료합니다. 먼저 n을 출력한 다음, countdown(n-1)을 호출합니다.*

해설

```
def countdown(n):
    # 기저 조건: n이 0 이하이면 종료
    if n <= 0:
        return
    # 현재 숫자를 출력
    print(n)
    # 재귀 호출: n-1부터 카운트다운
    countdown(n - 1)

# 동작 과정:
# countdown(5): print(5) → countdown(4)
# countdown(4): print(4) → countdown(3)
# countdown(3): print(3) → countdown(2)
# countdown(2): print(2) → countdown(1)
# countdown(1): print(1) → countdown(0)
# countdown(0): 종료
```

중급 재귀 맛보기 (Expert)

11. 재귀함수를 사용하여 콜라츠 추측 문제를 해결하세요. 규칙: n이 짝수이면 2로 나누고, 홀수이면 3을 곱하고 1을 더합니다. 1에 도달할 때까지 몇 번의 단계가 필요한지 계산하세요.

```
def collatz_steps(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(collatz_steps(1))   # 0이 나와야 합니다 (이미 1)
print(collatz_steps(2))   # 1이 나와야 합니다 (2→1)
print(collatz_steps(3))   # 7이 나와야 합니다 (3→10→5→16→8→4→2→1)
```

입력예시

```
3
```

출력예시

```
7
```

> **힌트:** *기저 조건: n이 1이면 0을 반환합니다. 재귀 호출: n이 짝수이면 1 + collatz_steps(n//2), 홀수이면 1 + collatz_steps(3*n+1)을 반환합니다.*

해설

```
def collatz_steps(n):
    # 기저 조건: n이 1이면 더 이상 단계가 필요 없음
    if n == 1:
        return 0

    # n이 짝수인 경우
    if n % 2 == 0:
        return 1 + collatz_steps(n // 2)
    # n이 홀수인 경우
    else:
        return 1 + collatz_steps(3 * n + 1)

# 과정을 보여주는 버전:
def collatz_sequence(n):
    """콜라츠 수열을 출력하면서 단계 수를 계산"""
    def helper(num, steps):
        print(num, end=" → " if num != 1 else "\n")
        if num == 1:
            return steps

        if num % 2 == 0:
            return helper(num // 2, steps + 1)
        else:
            return helper(3 * num + 1, steps + 1)

    return helper(n, 0)

# 동작 과정 예시:
# collatz_steps(3)
# → n=3 (홀수) → 1 + collatz_steps(3*3+1) = 1 + collatz_steps(10)
# → n=10 (짝수) → 1 + 1 + collatz_steps(10//2) = 2 + collatz_steps(5)
# → n=5 (홀수) → 2 + 1 + collatz_steps(3*5+1) = 3 + collatz_steps(16)
# → n=16 (짝수) → 3 + 1 + collatz_steps(16//2) = 4 + collatz_steps(8)
# → n=8 (짝수) → 4 + 1 + collatz_steps(8//2) = 5 + collatz_steps(4)
# → n=4 (짝수) → 5 + 1 + collatz_steps(4//2) = 6 + collatz_steps(2)
# → n=2 (짝수) → 6 + 1 + collatz_steps(2//2) = 7 + collatz_steps(1)
# → n=1 → 7 + 0 = 7

# 테스트
print("3의 콜라츠 수열:")
print(f"단계 수: {collatz_sequence(3)}")
```

12. 재귀함수를 사용하여 피보나치 수열의 n번째 값을 계산하는 함수를 작성하세요. 피보나치 수열: F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2) (n≥2) 예: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

```
def fibonacci(n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(fibonacci(0))   # 0이 나와야 합니다
print(fibonacci(1))   # 1이 나와야 합니다
print(fibonacci(6))   # 8이 나와야 합니다
print(fibonacci(10))  # 55가 나와야 합니다
```

입력예시

```
6
```

출력예시

```
8
```

> **힌트:** *기저 조건: n이 0이면 0, n이 1이면 1을 반환합니다. 재귀 호출: fibonacci(n-1) + fibonacci(n-2)를 반환합니다.*

해설

```
def fibonacci(n):
    # 기저 조건: F(0) = 0, F(1) = 1
    if n == 0:
        return 0
    if n == 1:
        return 1

    # 재귀 호출: F(n) = F(n-1) + F(n-2)
    return fibonacci(n - 1) + fibonacci(n - 2)

# 동작 과정 예시 (fibonacci(5)):
#                    fib(5)
#                   /      \
#               fib(4)    fib(3)
#               /    \    /     \
#           fib(3) fib(2) fib(2) fib(1)
#           /  \   /  \   /  \     |
#       fib(2) fib(1) fib(1) fib(0) fib(1) fib(0)  1
#       /  \     |     |     |     |     |
#   fib(1) fib(0) 1     1     0     1     0
#     |     |
#     1     0
#
# 결과: fib(5) = 5

# 참고: 이 방법은 같은 값을 여러 번 계산하므로 비효율적입니다.
# 실제로는 메모이제이션이나 동적 프로그래밍을 사용해야 합니다.
```

13. 재귀함수를 사용하여 리스트에서 가장 작은 값을 찾는 함수를 작성하세요. (분할정복 방식) 빈 리스트가 입력되지 않는다고 가정합니다.

```
def find_min(arr):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(find_min([3, 7, 1, 9, 2]))  # 1이 나와야 합니다
print(find_min([5]))              # 5가 나와야 합니다
```

입력예시

```
[3, 7, 1, 9, 2]
```

출력예시

```
1
```

> **힌트:** *기저 조건: 리스트에 원소가 1개면 그 원소를 반환합니다. 재귀 호출: 첫 번째 원소와 나머지 원소들 중 최솟값을 비교하여 더 작은 값을 반환합니다.*

해설

```
def find_min(arr):
    # 기저 조건: 리스트에 원소가 1개면 그 원소가 최솟값
    if len(arr) == 1:
        return arr[0]

    # 재귀 호출: 나머지 원소들 중 최솟값 찾기
    min_of_rest = find_min(arr[1:])

    # 첫 번째 원소와 나머지 중 최솟값을 비교
    return min(arr[0], min_of_rest)

# 더 간단한 버전:
def find_min_simple(arr):
    if len(arr) == 1:
        return arr[0]
    return min(arr[0], find_min_simple(arr[1:]))

# 동작 과정 예시:
# find_min([3,7,1,9,2]) = min(3, find_min([7,1,9,2]))
#                       = min(3, min(7, find_min([1,9,2])))
#                       = min(3, min(7, min(1, find_min([9,2]))))
#                       = min(3, min(7, min(1, min(9, find_min([2])))))
#                       = min(3, min(7, min(1, min(9, 2))))
#                       = min(3, min(7, min(1, 2)))
#                       = min(3, min(7, 1))
#                       = min(3, 1)
#                       = 1
```

14. 재귀함수를 사용하여 리스트에서 특정 값이 몇 개 있는지 세는 함수를 작성하세요. 예: [1, 2, 1, 3, 1, 4]에서 1은 3개

```
def count_value(arr, target):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(count_value([1, 2, 1, 3, 1, 4], 1))  # 3이 나와야 합니다
print(count_value([5, 5, 5, 2], 5))        # 3이 나와야 합니다
```

입력예시

```
arr=[1, 2, 1, 3, 1, 4], target=1
```

출력예시

```
3
```

> **힌트:** *기저 조건: 리스트가 비어있으면 0을 반환합니다. 재귀 호출: 첫 번째 원소가 target과 같으면 1을 더하고, 그렇지 않으면 0을 더한 다음 count_value(arr[1:], target)을 호출합니다.*

해설

```
def count_value(arr, target):
    # 기저 조건: 빈 리스트이면 0 반환
    if not arr:
        return 0

    # 첫 번째 원소가 찾는 값과 같은지 확인
    if arr[0] == target:
        # 같으면 1을 더하고 나머지 리스트 확인
        return 1 + count_value(arr[1:], target)
    else:
        # 다르면 0을 더하고 나머지 리스트 확인
        return 0 + count_value(arr[1:], target)

# 더 간단한 버전:
def count_value_simple(arr, target):
    if not arr:
        return 0
    return (1 if arr[0] == target else 0) + count_value_simple(arr[1:], target)

# 동작 과정 예시:
# count_value([1,2,1,3,1,4], 1) = 1 + count_value([2,1,3,1,4], 1)  # 1==1
#                               = 1 + 0 + count_value([1,3,1,4], 1)   # 2!=1
#                               = 1 + 0 + 1 + count_value([3,1,4], 1)  # 1==1
#                               = 1 + 0 + 1 + 0 + count_value([1,4], 1)    # 3!=1
#                               = 1 + 0 + 1 + 0 + 1 + count_value([4], 1)  # 1==1
#                               = 1 + 0 + 1 + 0 + 1 + 0 + count_value([], 1)   # 4!=1
#                               = 1 + 0 + 1 + 0 + 1 + 0 + 0
#                               = 3
```