재귀함수 기초 연습 - Easy

기초 재귀 연습 (Apprentice)

1. 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
```

2. 재귀함수를 사용하여 1부터 n까지의 합을 계산하는 함수를 작성하세요. 합은 1 + 2 + 3 + ... + n 입니다. 예: n=5일 때 1+2+3+4+5 = 15

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

# 테스트
print(sum_to_n(5))  # 15가 나와야 합니다
print(sum_to_n(10)) # 55가 나와야 합니다
```

입력예시

```
5
```

출력예시

```
15
```

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

해설

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

# 동작 과정 예시:
# sum_to_n(5) = 5 + sum_to_n(4)
#             = 5 + 4 + sum_to_n(3)
#             = 5 + 4 + 3 + sum_to_n(2)
#             = 5 + 4 + 3 + 2 + sum_to_n(1)
#             = 5 + 4 + 3 + 2 + 1 + sum_to_n(0)
#             = 5 + 4 + 3 + 2 + 1 + 0
#             = 15
```

3. 재귀함수를 사용하여 a의 n제곱(a^n)을 계산하는 함수를 작성하세요. 거듭제곱은 a^n = a × a × ... × a (n번 곱하기) 입니다. 예: 2^4 = 2 × 2 × 2 × 2 = 16

```
def power(a, n):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(power(2, 4))  # 16이 나와야 합니다
print(power(3, 3))  # 27이 나와야 합니다
```

입력예시

```
a = 2, n = 4
```

출력예시

```
16
```

> **힌트:** *기저 조건: n이 0이면 1을 반환합니다. 재귀 호출: a * power(a, n-1)을 반환합니다.*

해설

```
def power(a, n):
    # 기저 조건: a^0 = 1
    if n == 0:
        return 1
    # 재귀 호출: a * a^(n-1)
    return a * power(a, n - 1)

# 동작 과정 예시:
# power(2, 4) = 2 * power(2, 3)
#             = 2 * 2 * power(2, 2)
#             = 2 * 2 * 2 * power(2, 1)
#             = 2 * 2 * 2 * 2 * power(2, 0)
#             = 2 * 2 * 2 * 2 * 1
#             = 16
```

4. 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('****')
```

5. 재귀함수를 사용하여 문자열의 길이를 계산하는 함수를 작성하세요. (len() 함수를 사용하지 않고) 예: "hello"의 길이는 5

```
def string_length(s):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(string_length("hello"))   # 5가 나와야 합니다
print(string_length("python"))  # 6이 나와야 합니다
print(string_length(""))        # 0이 나와야 합니다
```

입력예시

```
"hello"
```

출력예시

```
5
```

> **힌트:** *기저 조건: 문자열이 비어있으면 0을 반환합니다. 재귀 호출: 1 + string_length(s[1:])을 반환합니다.*

해설

```
def string_length(s):
    # 기저 조건: 빈 문자열이면 0 반환
    if not s:  # s == "" 와 같음
        return 0
    # 재귀 호출: 1 + (첫 글자를 제거한 문자열의 길이)
    return 1 + string_length(s[1:])

# 동작 과정 예시:
# string_length("hello") = 1 + string_length("ello")
#                        = 1 + 1 + string_length("llo")
#                        = 1 + 1 + 1 + string_length("lo")
#                        = 1 + 1 + 1 + 1 + string_length("o")
#                        = 1 + 1 + 1 + 1 + 1 + string_length("")
#                        = 1 + 1 + 1 + 1 + 1 + 0
#                        = 5
```

6. 재귀함수를 사용하여 숫자 삼각형을 출력하는 함수를 작성하세요. i번째 줄에는 숫자 i가 i개 출력됩니다. 예: n=4일 때 1, 22, 333, 4444

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

# 테스트
print_number_triangle(4)
```

입력예시

```
4
```

출력예시

```
1
22
333
4444
```

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

해설

```
def print_number_triangle(n):
    # 기저 조건: n이 0 이하이면 종료
    if n <= 0:
        return
    # 재귀 호출: 먼저 n-1줄까지 출력
    print_number_triangle(n - 1)
    # 현재 줄에 숫자 n을 n개 출력
    print(str(n) * n)

# 동작 과정:
# print_number_triangle(4) → print_number_triangle(3) → ... → print_number_triangle(0)
# print_number_triangle(0): 종료
# print_number_triangle(1): print('1')
# print_number_triangle(2): print('22')
# print_number_triangle(3): print('333')
# print_number_triangle(4): print('4444')
```

7. 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)
```

8. 재귀함수를 사용하여 별(*)로 역삼각형을 출력하는 함수를 작성하세요. n줄짜리 역삼각형에서 첫 번째 줄에 n개, 마지막 줄에 1개의 별을 출력합니다.

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

# 테스트
print_reverse_stars(4)
```

입력예시

```
4
```

출력예시

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

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

해설

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

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

9. 재귀함수를 사용하여 주어진 양의 정수의 각 자릿수를 모두 더한 값을 계산하는 함수를 작성하세요. 예: 1234의 경우 1+2+3+4 = 10

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

# 테스트
print(sum_of_digits(1234))  # 10이 나와야 합니다
print(sum_of_digits(987))   # 24가 나와야 합니다
```

입력예시

```
1234
```

출력예시

```
10
```

> **힌트:** *기저 조건: n이 0이면 0을 반환합니다. 재귀 호출: (n%10) + sum_of_digits(n//10)을 반환합니다. n%10은 마지막 자릿수, n//10은 마지막 자릿수를 제거한 숫자입니다.*

해설

```
def sum_of_digits(n):
    # 기저 조건: n이 0이면 0 반환
    if n == 0:
        return 0
    # 재귀 호출: 마지막 자릿수 + 나머지 자릿수들의 합
    return (n % 10) + sum_of_digits(n // 10)

# 동작 과정 예시:
# sum_of_digits(1234) = 4 + sum_of_digits(123)
#                     = 4 + 3 + sum_of_digits(12)
#                     = 4 + 3 + 2 + sum_of_digits(1)
#                     = 4 + 3 + 2 + 1 + sum_of_digits(0)
#                     = 4 + 3 + 2 + 1 + 0
#                     = 10
```

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 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
```

12. 재귀함수를 사용하여 리스트에서 특정 값이 처음 나타나는 인덱스를 찾는 함수를 작성하세요. 값이 없으면 -1을 반환합니다. (선형 탐색의 재귀 구현)

```
def find_index(arr, target, index=0):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(find_index([1, 3, 5, 7, 3], 3))   # 1이 나와야 합니다 (첫 번째 3)
print(find_index([1, 2, 3, 4], 5))      # -1이 나와야 합니다
print(find_index([1, 2, 3, 4], 4))      # 3이 나와야 합니다
```

입력예시

```
arr=[1,3,5,7,3], target=3
```

출력예시

```
1
```

> **힌트:** *기저 조건: index가 배열 크기 이상이면 -1을 반환합니다. arr[index]가 target과 같으면 index를 반환하고, 다르면 find_index(arr, target, index+1)을 호출합니다.*

해설

```
def find_index(arr, target, index=0):
    # 기저 조건: 인덱스가 배열 범위를 벗어나면 찾지 못함
    if index >= len(arr):
        return -1

    # 현재 위치의 값이 target과 같은지 확인
    if arr[index] == target:
        return index

    # 다음 인덱스를 재귀적으로 탐색
    return find_index(arr, target, index + 1)

# 슬라이싱을 사용한 버전:
def find_index_slice(arr, target):
    # 빈 배열이면 찾지 못함
    if not arr:
        return -1

    # 첫 번째 원소가 target과 같으면 0 반환
    if arr[0] == target:
        return 0

    # 나머지 부분에서 재귀 탐색
    rest_index = find_index_slice(arr[1:], target)

    # 나머지 부분에서 찾지 못했으면 -1
    if rest_index == -1:
        return -1

    # 나머지 부분에서 찾았으면 인덱스에 1을 더함
    return rest_index + 1

# 동작 과정 예시:
# find_index([1,3,5,7,3], 3, 0)
# → arr[0]=1 != 3 → find_index([1,3,5,7,3], 3, 1)
# → arr[1]=3 == 3 → return 1

# 마지막 등장 인덱스를 찾고 싶다면:
def find_last_index(arr, target, index=None):
    if index is None:
        index = len(arr) - 1

    if index < 0:
        return -1

    if arr[index] == target:
        return index

    return find_last_index(arr, target, index - 1)
```

13. 재귀함수를 사용하여 주어진 문자열이 회문(앞뒤가 같은 문자열)인지 확인하는 함수를 작성하세요. 예: "racecar"는 회문, "hello"는 회문이 아님

```
def is_palindrome(s):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(is_palindrome("racecar"))  # True가 나와야 합니다
print(is_palindrome("hello"))    # False가 나와야 합니다
print(is_palindrome("a"))        # True가 나와야 합니다
print(is_palindrome(""))         # True가 나와야 합니다
```

입력예시

```
"racecar"
```

출력예시

```
True
```

> **힌트:** *기저 조건: 문자열 길이가 0 또는 1이면 True를 반환합니다. 재귀 호출: 첫 글자와 마지막 글자가 같으면 가운데 부분을 재귀적으로 검사하고, 다르면 False를 반환합니다.*

해설

```
def is_palindrome(s):
    # 기저 조건: 빈 문자열이거나 한 글자이면 회문
    if len(s) <= 1:
        return True

    # 첫 글자와 마지막 글자 비교
    if s[0] != s[-1]:
        return False

    # 첫 글자와 마지막 글자를 제거한 가운데 부분을 재귀적으로 검사
    return is_palindrome(s[1:-1])

# 동작 과정 예시:
# is_palindrome("racecar")
# → s[0]='r', s[-1]='r' 같음 → is_palindrome("aceca")
# → s[0]='a', s[-1]='a' 같음 → is_palindrome("cec")
# → s[0]='c', s[-1]='c' 같음 → is_palindrome("e")
# → len("e") == 1 → True

# is_palindrome("hello")
# → s[0]='h', s[-1]='o' 다름 → False

# 대소문자를 무시하고 싶다면:
def is_palindrome_case_insensitive(s):
    s = s.lower()  # 모두 소문자로 변환
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return is_palindrome_case_insensitive(s[1:-1])
```

14. 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.
```