Exercise. 재귀함수 종합 연습

Section 1 - 재귀함수 기본 이해

1. 전역변수와 재귀함수를 사용하여 특정 문자를 n번 출력하는 함수를 작성하세요. 전역변수 count를 사용하여 현재까지 출력한 개수를 추적합니다.

```
count = 0  # 전역변수

def print_char_with_global(char, n):
    global count
    # 여기에 코드를 작성하세요
    pass

def reset_count():
    global count
    count = 0

# 테스트
reset_count()
print("전역변수 사용:")
print_char_with_global('@', 4)  # @@@@ 가 출력되어야 합니다
print(f"\n총 {count}개 출력됨")
```

입력예시

```
char='@', n=4
```

출력예시

```
@@@@
총 4개 출력됨
```

> **힌트:** *기저 조건: count가 n과 같아지면 종료합니다. 재귀 호출: 문자를 출력하고 count를 1 증가시킨 다음, 함수를 재귀 호출합니다.*

해설

```
count = 0  # 전역변수

def print_char_with_global(char, n):
    global count

    # 기저 조건: count가 n에 도달하면 종료
    if count >= n:
        return

    # 문자 출력
    print(char, end='')

    # 전역변수 증가
    count += 1

    # 재귀 호출
    print_char_with_global(char, n)

def reset_count():
    global count
    count = 0

# 동작 과정 예시:
# reset_count() → count = 0
# print_char_with_global('@', 4)
# count=0: print('@'), count=1 → 재귀 호출
# count=1: print('@'), count=2 → 재귀 호출
# count=2: print('@'), count=3 → 재귀 호출
# count=3: print('@'), count=4 → 재귀 호출
# count=4: count >= n이므로 종료
# 결과: @@@@

# 참고: 전역변수 사용은 일반적으로 권장되지 않습니다.
# 함수의 순수성을 해치고 부작용을 일으킬 수 있기 때문입니다.
```

Section 2 - 재귀함수를 제어문으로 변환

2. 다음 재귀함수를 제어문만 사용하여 하나의 함수로 구현하세요.

```
def kiwi_recursive(n):
    if n == 0:
        return "0"
    if n == 1:
        return "1"
    return kiwi_recursive(n // 2) + str(n % 2)

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

입력예시

```
10
```

출력예시

```
1010
```

해설

```
def kiwi(n):
    if n == 0:
        return "0"

    result = ""
    while n > 0:
        result = str(n % 2) + result
        n = n // 2
    return result
```

Section 3 - 제어문을 재귀함수로 변환

3. 다음 제어문으로 구현된 함수를 재귀함수로 변환하세요.

```
def fig_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

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

입력예시

```
5
```

출력예시

```
120
```

해설

```
def fig(n):
    if n <= 1:
        return 1
    return n * fig(n - 1)
```

Exercise. 재귀함수 종합 연습

Section 1 - 재귀함수 기본 이해

1. 재귀함수를 사용하여 10진수를 임의의 n진수 문자열로 변환하는 함수를 작성하세요. 2진수부터 16진수까지 지원하며, 10 이상의 숫자는 A, B, C, D, E, F로 표현합니다. 예: decimal_to_base(255, 16) → "FF"

```
def decimal_to_base(n, base):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(f"decimal_to_base(10, 2) = {decimal_to_base(10, 2)}")    # "1010"
print(f"decimal_to_base(255, 16) = {decimal_to_base(255, 16)}")  # "FF"
print(f"decimal_to_base(100, 8) = {decimal_to_base(100, 8)}")   # "144"
print(f"decimal_to_base(15, 16) = {decimal_to_base(15, 16)}")   # "F"
```

입력예시

```
n=255, base=16
```

출력예시

```
"FF"
```

> **힌트:** *기저 조건: n이 0이면 "0"을, n < base이면 해당 자릿값을 반환합니다. 10 이상의 값은 'A', 'B' 등으로 변환해야 합니다. 재귀 호출: decimal_to_base(n//base, base) + 나머지에 해당하는 문자를 반환합니다.*

해설

```
def decimal_to_base(n, base):
    # 기저 조건: n이 0이면 "0" 반환
    if n == 0:
        return "0"

    # 기저 조건: n이 base보다 작으면 해당 문자 반환
    if n < base:
        if n < 10:
            return str(n)
        else:
            # 10, 11, 12, 13, 14, 15를 A, B, C, D, E, F로 변환
            return chr(ord('A') + n - 10)

    # 나머지를 문자로 변환
    remainder = n % base
    if remainder < 10:
        remainder_char = str(remainder)
    else:
        remainder_char = chr(ord('A') + remainder - 10)

    # 재귀 호출: 몫의 base진수 + 나머지 문자
    return decimal_to_base(n // base, base) + remainder_char

# 동작 과정 예시:
# decimal_to_base(255, 16)
# = decimal_to_base(255//16, 16) + 'F'  (255%16=15 → 'F')
# = decimal_to_base(15, 16) + 'F'
# = "F" + "F"  (15는 base보다 작으므로 기저 조건, 15 → 'F')
# = "FF"

# decimal_to_base(100, 8)
# = decimal_to_base(100//8, 8) + '4'  (100%8=4 → '4')
# = decimal_to_base(12, 8) + '4'
# = (decimal_to_base(12//8, 8) + '4') + '4'  (12%8=4 → '4')
# = (decimal_to_base(1, 8) + '4') + '4'
# = ("1" + "4") + "4"  (1은 base보다 작음)
# = "14" + "4"
# = "144"

# 진법 변환의 일반 원리:
# n진법으로 변환하려면 계속 n으로 나누면서
# 나머지를 역순으로 연결하면 됩니다.
```

Section 2 - 재귀함수를 제어문으로 변환

2. 다음 재귀함수를 제어문만 사용하여 하나의 함수로 구현하세요.

```
def banana_recursive(n):
    if n <= 0:
        return
    banana_recursive(n - 1)
    print(n)

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

입력예시

```
5
```

출력예시

```
1
2
3
4
5
```

해설

```
def banana(n):
    for i in range(1, n + 1):
        print(i)
```

Section 3 - 제어문을 재귀함수로 변환

3. 다음 제어문으로 구현된 함수를 재귀함수로 변환하세요.

```
def honeydew_iterative(a, b):
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a

def honeydew(a, b):
    # 여기에 코드를 작성하세요
    pass
```

입력예시

```
a=48, b=18
```

출력예시

```
6
```

해설

```
def honeydew(a, b):
    if b == 0:
        return a
    return honeydew(b, a % b)
```

Exercise. 재귀함수 종합 연습

Section 1 - 재귀함수 기본 이해

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

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

# 테스트
print("숫자 삼각형 (5줄):")
number_triangle(5)
```

입력예시

```
5
```

출력예시

```
1
22
333
4444
55555
```

> **힌트:** *기저 조건: n이 0 이하이면 종료합니다. 재귀 호출: 먼저 number_triangle(n-1)을 호출한 다음, 숫자 n을 n개 출력합니다.*

해설

```
def number_triangle(n):
    # 기저 조건: n이 0 이하이면 종료
    if n <= 0:
        return

    # 재귀 호출: 먼저 n-1줄까지 출력
    number_triangle(n - 1)

    # 현재 줄에 숫자 n을 n개 출력
    print(str(n) * n)

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

Section 2 - 재귀함수를 제어문으로 변환

2. 다음 재귀함수를 제어문만 사용하여 하나의 함수로 구현하세요.

```
def lemon_recursive(n, base):
    if n == 0:
        return "0"
    if n < base:
        if n < 10:
            return str(n)
        else:
            return chr(ord('A') + n - 10)

    remainder = n % base
    if remainder < 10:
        remainder_char = str(remainder)
    else:
        remainder_char = chr(ord('A') + remainder - 10)

    return lemon_recursive(n // base, base) + remainder_char

def lemon(n, base):
    # 여기에 코드를 작성하세요
    pass
```

입력예시

```
n=255, base=16
```

출력예시

```
FF
```

해설

```
def lemon(n, base):
    if n == 0:
        return "0"

    result = ""
    while n > 0:
        remainder = n % base
        if remainder < 10:
            result = str(remainder) + result
        else:
            result = chr(ord('A') + remainder - 10) + result
        n = n // base
    return result
```

Section 3 - 제어문을 재귀함수로 변환

3. 다음 제어문으로 구현된 함수를 재귀함수로 변환하세요.

```
def lemon_iterative(base, exponent):
    result = 1
    for i in range(exponent):
        result *= base
    return result

def lemon(base, exponent):
    # 여기에 코드를 작성하세요
    pass
```

입력예시

```
base=2, exponent=5
```

출력예시

```
32
```

해설

```
def lemon(base, exponent):
    if exponent == 0:
        return 1
    return base * lemon(base, exponent - 1)
```

Exercise. 재귀함수 종합 연습

Section 1 - 재귀함수 기본 이해

1. 재귀함수를 사용하여 10진수를 2진수 문자열로 변환하는 함수를 작성하세요. 예: 10 → "1010", 7 → "111"

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

# 테스트
print(f"decimal_to_binary(10) = {decimal_to_binary(10)}")  # "1010"
print(f"decimal_to_binary(7) = {decimal_to_binary(7)}")    # "111"
print(f"decimal_to_binary(0) = {decimal_to_binary(0)}")    # "0"
print(f"decimal_to_binary(1) = {decimal_to_binary(1)}")    # "1"
```

입력예시

```
10
```

출력예시

```
"1010"
```

> **힌트:** *기저 조건: n이 0이면 빈 문자열을, n이 1이면 "1"을 반환합니다. 재귀 호출: decimal_to_binary(n//2) + str(n%2)를 반환합니다.*

해설

```
def decimal_to_binary(n):
    # 기저 조건: n이 0이면 "0" 반환 (특별한 경우)
    if n == 0:
        return "0"

    # 기저 조건: n이 1이면 "1" 반환
    if n == 1:
        return "1"

    # 재귀 호출: 몫의 2진수 + 나머지
    return decimal_to_binary(n // 2) + str(n % 2)

# 동작 과정 예시:
# decimal_to_binary(10)
# = decimal_to_binary(10//2) + str(10%2)
# = decimal_to_binary(5) + "0"
# = (decimal_to_binary(5//2) + str(5%2)) + "0"
# = (decimal_to_binary(2) + "1") + "0"
# = ((decimal_to_binary(2//2) + str(2%2)) + "1") + "0"
# = ((decimal_to_binary(1) + "0") + "1") + "0"
# = (("1" + "0") + "1") + "0"
# = ("10" + "1") + "0"
# = "101" + "0"
# = "1010"

# 진법 변환 원리:
# 10진수를 2진수로 변환하려면
# 계속 2로 나누면서 나머지를 역순으로 연결
```

Section 2 - 재귀함수를 제어문으로 변환

2. 다음 재귀함수를 제어문만 사용하여 하나의 함수로 구현하세요.

```
def elderberry_recursive(n):
    if n <= 0:
        return
    elderberry_recursive(n - 1)
    print(str(n) * n)

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

입력예시

```
5
```

출력예시

```
1
22
333
4444
55555
```

해설

```
def elderberry(n):
    for i in range(1, n + 1):
        print(str(i) * i)
```

Section 3 - 제어문을 재귀함수로 변환

3. 다음 제어문으로 구현된 함수를 재귀함수로 변환하세요.

```
def durian_iterative(n):
    for i in range(n, 0, -1):
        print('*' * i)

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

입력예시

```
5
```

출력예시

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

해설

```
def durian(n):
    if n <= 0:
        return
    print('*' * n)
    durian(n - 1)
```

Exercise. 재귀함수 종합 연습

Section 1 - 재귀함수 기본 이해

1. 재귀함수를 사용하여 10진수를 임의의 n진수 문자열로 변환하는 함수를 작성하세요. 2진수부터 16진수까지 지원하며, 10 이상의 숫자는 A, B, C, D, E, F로 표현합니다. 예: decimal_to_base(255, 16) → "FF"

```
def decimal_to_base(n, base):
    # 여기에 코드를 작성하세요
    pass

# 테스트
print(f"decimal_to_base(10, 2) = {decimal_to_base(10, 2)}")    # "1010"
print(f"decimal_to_base(255, 16) = {decimal_to_base(255, 16)}")  # "FF"
print(f"decimal_to_base(100, 8) = {decimal_to_base(100, 8)}")   # "144"
print(f"decimal_to_base(15, 16) = {decimal_to_base(15, 16)}")   # "F"
```

입력예시

```
n=255, base=16
```

출력예시

```
"FF"
```

> **힌트:** *기저 조건: n이 0이면 "0"을, n < base이면 해당 자릿값을 반환합니다. 10 이상의 값은 'A', 'B' 등으로 변환해야 합니다. 재귀 호출: decimal_to_base(n//base, base) + 나머지에 해당하는 문자를 반환합니다.*

해설

```
def decimal_to_base(n, base):
    # 기저 조건: n이 0이면 "0" 반환
    if n == 0:
        return "0"

    # 기저 조건: n이 base보다 작으면 해당 문자 반환
    if n < base:
        if n < 10:
            return str(n)
        else:
            # 10, 11, 12, 13, 14, 15를 A, B, C, D, E, F로 변환
            return chr(ord('A') + n - 10)

    # 나머지를 문자로 변환
    remainder = n % base
    if remainder < 10:
        remainder_char = str(remainder)
    else:
        remainder_char = chr(ord('A') + remainder - 10)

    # 재귀 호출: 몫의 base진수 + 나머지 문자
    return decimal_to_base(n // base, base) + remainder_char

# 동작 과정 예시:
# decimal_to_base(255, 16)
# = decimal_to_base(255//16, 16) + 'F'  (255%16=15 → 'F')
# = decimal_to_base(15, 16) + 'F'
# = "F" + "F"  (15는 base보다 작으므로 기저 조건, 15 → 'F')
# = "FF"

# decimal_to_base(100, 8)
# = decimal_to_base(100//8, 8) + '4'  (100%8=4 → '4')
# = decimal_to_base(12, 8) + '4'
# = (decimal_to_base(12//8, 8) + '4') + '4'  (12%8=4 → '4')
# = (decimal_to_base(1, 8) + '4') + '4'
# = ("1" + "4") + "4"  (1은 base보다 작음)
# = "14" + "4"
# = "144"

# 진법 변환의 일반 원리:
# n진법으로 변환하려면 계속 n으로 나누면서
# 나머지를 역순으로 연결하면 됩니다.
```

Section 2 - 재귀함수를 제어문으로 변환

2. 다음 재귀함수를 제어문만 사용하여 하나의 함수로 구현하세요.

```
def cherry_recursive(n, current=1):
    if current > n:
        return
    print('*' * current)
    cherry_recursive(n, current + 1)

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

입력예시

```
5
```

출력예시

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

해설

```
def cherry(n):
    for i in range(1, n + 1):
        print('*' * i)
```

Section 3 - 제어문을 재귀함수로 변환

3. 다음 제어문으로 구현된 함수를 재귀함수로 변환하세요.

```
def durian_iterative(n):
    for i in range(n, 0, -1):
        print('*' * i)

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

입력예시

```
5
```

출력예시

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

해설

```
def durian(n):
    if n <= 0:
        return
    print('*' * n)
    durian(n - 1)
```