Exercise - cnsh-121 Python Programming Problems (2024.09.14)

Python Programming Problems

1. 다음 유클리드 호제법 함수에서 while문의 조건식을 완성하세요.

```
def gcd(a, b):
    while ____:  # 여기를 채우세요
        a, b = b % a, a
    return b
```

입력예시

```
print(gcd(48, 18))
print(gcd(100, 25))
```

출력예시

```
6
25
```

해설

```
def gcd(a, b):
    while a != 0:  # 정답: a != 0
        a, b = b % a, a
    return b
```

2. 다음 재귀 합계 함수와 같은 기능을 하는 for문 기반 함수를 구현하세요.

```
def sum(n):
    if n == 1:
        return 1
    return n + sum(n-1)

# for문 버전을 작성하세요
def sum_for(n):
    pass
```

입력예시

```
print(sum_for(5))
print(sum_for(4))
```

출력예시

```
15
10
```

해설

```
def sum_for(n):
    result = 0
    for i in range(1, n+1):
        result += i
    return result
```

3. 다음 Movie 클래스를 분석하고, 주어진 코드를 실행했을 때의 출력 결과를 예측하세요.

```
class Movie:
    def __init__(self, name, seat):
        self.name = name
        self.total_seats = seat

    def reserve(self):
        if self.total_seats > 0:
            self.total_seats -= 1
            return True
        else:
            return False
```

입력예시

```
seat1 = Movie('ET', 5)
result = seat1.reserve()
print('예약 성공' if result else '예약 실패')

# 추가 예약 시도
for i in range(6):
    result = seat1.reserve()
    print(f"{i+2}번째 예약: {'성공' if result else '실패'}")
```

출력예시

```
예약 성공
2번째 예약: 성공
3번째 예약: 성공
4번째 예약: 성공
5번째 예약: 성공
6번째 예약: 실패
7번째 예약: 실패
```

해설

```
class Movie:
    def __init__(self, name, seat):
        self.name = name
        self.total_seats = seat

    def reserve(self):
        if self.total_seats > 0:
            self.total_seats -= 1
            return True
        else:
            return False

# 처음 total_seats가 5이므로 5번까지는 예약이 가능하고,
# 6번째부터는 좌석이 없어서 예약 실패가 됩니다.
```

4. 다음 최소공배수 재귀함수에서 return 부분을 완성하세요.

```
def lcm(a, b):
    global n, m
    if a == b:
        return a
    if a > b:
        return ____  # 여기를 채우세요
    if a < b:
        return lcm(a+m, b)
```

입력예시

```
n, m = 3, 8
print(lcm(m, n))
```

출력예시

```
24
```

해설

```
def lcm(a, b):
    global n, m
    if a == b:
        return a
    if a > b:
        return lcm(a, b+n)  # 정답: lcm(a, b+n)
    if a < b:
        return lcm(a+m, b)
```

5. Distance 클래스를 사용해서 점들 (2,3), (3,5), (5,6), (-1,2) 사이의 모든 거리를 구하여 그 중 가장 먼 거리를 소수점 2자리로 출력하는 함수를 작성하세요.

```
import math

class Distance:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    def distance(self):
        return math.sqrt((self.x2 - self.x1) ** 2 + (self.y2 - self.y1) ** 2)

# 여기에 함수를 작성하세요
def find_max_distance():
    pass
```

입력예시

```
print(f"가장 먼 거리: {find_max_distance():.2f}")
```

출력예시

```
가장 먼 거리: 6.40
```

해설

```
import math

class Distance:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    def distance(self):
        return math.sqrt((self.x2 - self.x1) ** 2 + (self.y2 - self.y1) ** 2)

def find_max_distance():
    points = [(2, 3), (3, 5), (5, 6), (-1, 2)]
    max_dist = 0

    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            x1, y1 = points[i]
            x2, y2 = points[j]
            dist = Distance(x1, y1, x2, y2).distance()
            if dist > max_dist:
                max_dist = dist

    return max_dist
```

6. 다음 재귀 팩토리얼 함수와 같은 기능을 하는 while문 기반 함수를 구현하세요.

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

# while문 버전을 작성하세요
def factorial_while(n):
    pass
```

입력예시

```
print(factorial_while(5))
print(factorial_while(3))
```

출력예시

```
120
6
```

해설

```
def factorial_while(n):
    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result
```

7. 다음 카운트다운 함수에서 range() 파라메터를 완성하여 카운트다운이 되도록 하세요.

```
def countdown(n):
    for i in range(____, ____, ____):  # 여기를 채우세요
        print(i)
```

입력예시

```
countdown(5)
```

출력예시

```
5
4
3
2
1
```

해설

```
def countdown(n):
    for i in range(n, 0, -1):  # 정답: n, 0, -1
        print(i)
```

8. factorial(4)를 호출했을 때 재귀함수의 call stack이 어떻게 쌓이고 해제되는지 단계별로 서술하세요.

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

입력예시

```
factorial(4)
```

해설

```
# factorial(4) call stack 과정:
# 1. factorial(4) 호출 -> 4 * factorial(3) 대기
# 2. factorial(3) 호출 -> 3 * factorial(2) 대기
# 3. factorial(2) 호출 -> 2 * factorial(1) 대기
# 4. factorial(1) 호출 -> 1 반환 (종료조건)
# 5. factorial(2) = 2 * 1 = 2 반환
# 6. factorial(3) = 3 * 2 = 6 반환
# 7. factorial(4) = 4 * 6 = 24 반환

# Stack: factorial(4) -> factorial(3) -> factorial(2) -> factorial(1)
# 결과: 24 <- 6 <- 2 <- 1
```

9. 다음 재귀 최소공배수 함수를 while문으로 재구현하세요.

```
def lcm(a, b):
    global n, m
    if a == b:
        return a
    if a > b:
        return lcm(a, b+n)
    if a < b:
        return lcm(a+m, b)

# while문 버전을 작성하세요
def lcm_while(a, b):
    global n, m
    pass
```

입력예시

```
n, m = 4, 6
print(lcm_while(4, 6))
```

출력예시

```
12
```

해설

```
def lcm_while(a, b):
    global n, m
    while a != b:
        if a > b:
            b += n
        else:
            a += m
    return a
```

10. 다음 재귀 팩토리얼 함수와 같은 기능을 하는 for문 기반 함수를 구현하세요.

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

# for문 버전을 작성하세요
def factorial_for(n):
    pass
```

입력예시

```
print(factorial_for(5))
print(factorial_for(4))
```

출력예시

```
120
24
```

해설

```
def factorial_for(n):
    result = 1
    for i in range(n, 0, -1):
        result *= i
    return result
```

11. 다음 재귀 팩토리얼 함수에서 종료조건을 완성하세요.

```
def factorial(n):
    if ____:  # 여기를 채우세요
        return 1
    return n * factorial(n-1)
```

입력예시

```
print(factorial(5))
print(factorial(1))
```

출력예시

```
120
1
```

해설

```
def factorial(n):
    if n == 1:  # 정답: n == 1
        return 1
    return n * factorial(n-1)
```

12. 다음 코드에서 for문 내부를 구현하여 인접리스트를 만들도록 완성하세요.

```
V, E = map(int, input().split())
adj = [[] for _ in range(V)]

for i in range(E):
    # 여기에 인접리스트를 만드는 코드를 작성하세요
    pass

print(adj)
```

입력예시

```
4 3
0 1
2 0
3 2
```

출력예시

```
[[1, 2], [0], [0, 3], [2]]
```

해설

```
V, E = map(int, input().split())
adj = [[] for _ in range(V)]

for i in range(E):
    u, v = map(int, input().split())  # 정답
    adj[u].append(v)                  # 정답
    adj[v].append(u)                  # 정답

print(adj)
```

13. 다음 짝수합 함수에서 계산 로직 한 줄을 완성하세요.

```
def even_sum(n):
    result = 0
    for i in range(n+1):
        if i % 2 == 0:
            # 여기에 한 줄을 작성하세요
    return result
```

입력예시

```
print(even_sum(8))
print(even_sum(4))
```

출력예시

```
20
6
```

해설

```
def even_sum(n):
    result = 0
    for i in range(n+1):
        if i % 2 == 0:
            result += i  # 정답: result += i
    return result
```

14. 짝수합을 재귀함수로 구현하세요.

```
def even_sum(n):
    result = 0
    for i in range(n+1):
        if i % 2 == 0:
            result += i
    return result

# 재귀함수 버전을 작성하세요
def even_sum_recursive(n):
    pass
```

입력예시

```
print(even_sum_recursive(10))
print(even_sum_recursive(4))
```

출력예시

```
30
6
```

해설

```
def even_sum_recursive(n):
    if n < 0:
        return 0
    if n % 2 == 0:
        return n + even_sum_recursive(n-2)
    else:
        return even_sum_recursive(n-1)
```

15. 다음 while문 기반 최대공약수 함수에서 종료조건 두 줄을 완성하세요.

```
def gcd_while(a: int, b: int) -> int:
    a, b = abs(a), abs(b)
    # 여기에 종료조건 두 줄을 작성하세요

    while b != 0:
        a, b = b, a % b

    return a
```

입력예시

```
print(gcd_while(0, 25))
print(gcd_while(48, 0))
print(gcd_while(48, 18))
```

출력예시

```
25
48
6
```

해설

```
def gcd_while(a: int, b: int) -> int:
    a, b = abs(a), abs(b)
    if a == 0: return b  # 정답
    if b == 0: return a  # 정답

    while b != 0:
        a, b = b, a % b

    return a
```

16. q.empty()가 True일 때와 False일 때는 각각 어떤 상황인지 서술하세요.

```
import queue

q = queue.Queue()
for i in range(1, 4):
    q.put(i)
while not q.empty():
    print(q.get(), sep=' ')
```

해설

```
# q.empty()가 True일 때:
# - 큐에 아무 데이터도 없는 상태
# - 모든 데이터를 get()으로 꺼내서 큐가 비어있을 때
# - while 루프가 종료되는 조건

# q.empty()가 False일 때:
# - 큐에 하나 이상의 데이터가 있는 상태
# - put()으로 데이터를 넣었거나, 아직 get()으로 모두 꺼내지 않은 상태
# - while 루프가 계속 실행되는 조건
```

17. 다음 재귀함수에서 예상 출력대로 올바르게 동작하도록 range() 파라메터를 완성하세요.

```
def numtriangle(n):
    if n < 1:
        return
    numtriangle(n-1)

    for i in range(____):  # 여기를 채우세요
        print(i, end=' ')

    print()
```

입력예시

```
numtriangle(3)
```

출력예시

```
1
1 2
1 2 3
```

해설

```
def numtriangle(n):
    if n < 1:
        return
    numtriangle(n-1)

    for i in range(1, n+1):  # 정답: 1, n+1
        print(i, end=' ')

    print()
```

18. 다음 재귀함수와 같은 기능을 하는 while문 기반 함수를 구현하세요.

```
# 참고: 재귀함수 버전
def gcd_recursive(a: int, b: int) -> int:
    a, b = abs(a), abs(b)
    if b == 0:
        return a
    return gcd_recursive(b, a % b)

# 여기에 while문 버전을 작성하세요
def gcd_while(a: int, b: int) -> int:
    pass
```

입력예시

```
print(gcd_while(48, 18))
print(gcd_while(100, 25))
```

출력예시

```
6
25
```

해설

```
def gcd_while(a: int, b: int) -> int:
    a, b = abs(a), abs(b)
    if a == 0: return b
    if b == 0: return a

    while b != 0:
        a, b = b, a % b

    return a
```

19. 다음 Distance 클래스의 distance() 메서드를 완성하여 두 점 사이의 거리가 계산되도록 하세요.

```
import math

class Distance:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    def distance(self):
        # 두 점 사이의 거리를 계산하는 공식을 구현하세요
        pass
```

입력예시

```
distance1 = Distance(2, 3, 3, 5)
print(f"두 점 사이의 거리는 {distance1.distance():.2f}")
```

출력예시

```
두 점 사이의 거리는 2.24
```

해설

```
import math

class Distance:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    def distance(self):
        return math.sqrt((self.x2 - self.x1) ** 2 + (self.y2 - self.y1) ** 2)  # 정답
```

20. 다음 반복문 기반 카운트다운 함수와 같은 기능을 하는 재귀함수를 구현하세요.

```
def countdown(n):
    for i in range(n, 0, -1):
        print(i)

# 재귀함수 버전을 작성하세요
def countdown_recursive(n):
    pass
```

입력예시

```
countdown_recursive(4)
```

출력예시

```
4
3
2
1
```

해설

```
def countdown_recursive(n):
    if n < 1:
        return
    print(n)
    countdown_recursive(n-1)
```