<[파이썬 알고리즘 탐험] 제4장. 숫자들의 숨은 대장 찾기, 유클리드 호제법!>

선생님: (고대 그리스 두루마리 같은 것을 펼쳐 보이며) 얘들아, 안녕! 지난 시간에 우리가 두 숫자, 예를 들어 24와 10의 최대공약수를 찾을 때 어떻게 했었지?

코딩새싹 🌱: 24의 약수들을 쭉~ 적고, 10의 약수들도 쭉~ 적어서, 공통된 약수들 중에서 제일 큰 걸 골랐어요! 그래서 2가 나왔어요!

선생님: 맞아! 그렇게 해도 답을 찾을 수 있지만, 만약 숫자가 엄청나게 크다면 약수를 다 적는 것만 해도 하루 종일 걸릴지 몰라. 그런데 약 2300년 전, 유클리드라는 수학자가 "더 이상 나눌 수 없을 때까지 계속 나눠보면 어떨까?" 하는 기발한 생각을 했단다! 그게 바로 **유클리드 호제법(Euclidean Algorithm)**이야. '호제법'은 '서로(互) 나눈다(除)'는 뜻이지.

궁금이 💡: 서로 나눈다고요? 그게 어떻게 최대공약수를 찾는 방법이 돼요?

선생님: 아주 간단한 원리인데, 정말 신기하단다! 유클리드 아저씨는 이렇게 말했어.
"두 숫자 A와 B가 있을 때 (A가 B보다 크다고 해보자), **A와 B의 최대공약수는 B와 (A를 B로 나눈 나머지)의 최대공약수와 똑같다!**"

<1. 초콜릿 막대로 이해하는 유클리드 호제법!>

선생님: 자, 24칸짜리 긴 초콜릿 막대와 10칸짜리 짧은 초콜릿 막대가 있다고 상상해보자. 우리는 이 두 막대를 남김없이 똑같은 크기의 가장 큰 정사각형 조각으로 자르고 싶어. 그 정사각형 한 변의 길이가 바로 최대공약수가 되겠지?

1. **첫 번째 단계:** 긴 막대(24)를 짧은 막대(10)로 몇 번이나 자를 수 있을까?
   `24 = 10 * 2 + 4` (10칸짜리 두 조각이 나오고, 4칸이 남아!)
   유클리드 왈, "이제 24와 10의 문제는, 10과 4의 문제로 바뀌었도다!"
2. **두 번째 단계:** 그럼 이제 10칸짜리 막대와 4칸짜리 막대로 같은 작업을 반복!
   `10 = 4 * 2 + 2` (4칸짜리 두 조각이 나오고, 2칸이 남아!)
   유클리드 왈, "이제 10과 4의 문제는, 4와 2의 문제로 바뀌었도다!"
3. **세 번째 단계:** 다시 4칸짜리 막대와 2칸짜리 막대로!
   `4 = 2 * 2 + 0` (2칸짜리 두 조각이 나오고, 남는 게 없어! 나머지가 0!)

선생님: 나머지가 0이 되었을 때, 바로 그 직전에 나누었던 수(마지막에 남은 작은 막대 길이), 즉 **2**가 바로 24와 10의 최대공약수가 되는 거란다! 2칸짜리 정사각형 조각이 우리가 찾던 가장 큰 조각이었던 거지!

코딩새싹 🌱: 와! 계속해서 작은 숫자랑 나머지로 문제를 바꿔가니까 결국 답이 나오네요! 정말 신기한 방법이에요!

<2. 유클리드 아저씨의 지혜를 코드로! - 반복문 버전>

선생님: 이 멋진 방법을 파이썬 코드로 옮겨보자. while 반복문을 사용해서 나머지가 0이 될 때까지 계속하는 거야.

```
# 최대공약수 구하는 방법 - 유클리드 호제법 (반복문 사용)
def gcd(a, b): # a가 큰 수, b가 작은 수라고 생각해보자
  print(f"처음: a = {a}, b = {b}")
  while b != 0: # b가 0이 될 때까지 (나머지가 0이 될 때까지) 계속 반복!
    # (이전 a를 이전 b로 나눈 나머지)를 새로운 b로, (이전 b)를 새로운 a로!
    # a, b = b, a % b  <-- 이 한 줄이 마법의 핵심!

    remainder = a % b # 1. 이전 a를 이전 b로 나눈 나머지를 구하고
    a = b             # 2. 이전 b가 새로운 a가 되고 (더 큰 역할을 하던 애가 작은 애로)
    b = remainder     # 3. 나머지가 새로운 b가 된다! (다음엔 이 작은 애로 나눌 거야)
    print(f"변신! a = {a}, b = {b}")

  return a # b가 0이 되었을 때, a에 남아있는 값이 바로 최대공약수!

result = gcd(24, 10)
print("최대공약수는:", result)
```

궁금이 💡: a, b = b, a % b 이 한 줄이 "새로운 a는 이전 b가 되고, 새로운 b는 이전 a를 이전 b로 나눈 나머지가 된다"는 뜻이었군요! 변수 두 개가 한꺼번에 뿅 하고 바뀌는 마법 같아요!

선생님: 맞아! 파이썬은 이렇게 여러 변수의 값을 한 줄로 바꿀 수 있는 편리한 기능이 있단다. 저 while문은 b가 0이 되면 멈추고, 그때의 a 값을 돌려주지. 그게 바로 우리가 초콜릿 막대에서 찾았던 마지막 조각의 크기, 즉 최대공약수인 거야!

<3. 유클리드 아저씨의 지혜를 코드로! - 재귀 함수 버전>

선생님: 유클리드 호제법은 함수가 자기 자신을 다시 부르는 **재귀 함수**로도 멋지게 표현할 수 있어. 마치 거울

속의 거울처럼, 문제가 해결될 때까지 함수가 계속 자기 자신에게 조금 더 작은 문제를 풀어달라고 요청하는 거지.

```
# 유클리드 호제법 (재귀 함수 사용)
def gcd1(a, b):
  print(f"gcd1 함수 호출! a = {a}, b = {b}")
  if b == 0: # 가장 중요한 멈춤 조건! (나머지가 0이 되면)
    print(f"b가 0이 되었으므로, a={a}를 반환합니다.")
    return a # a가 바로 최대공약수!
  else: # 아직 나머지가 있다면,
    print(f"{b}와 {a % b}로 다시 gcd1 함수를 호출합니다.")
    return gcd1(b, a % b) # b를 새로운 a로, (a를 b로 나눈 나머지)를 새로운 b로 해서 다시 함수를 불러!


result_recursive = gcd1(24, 8)
print("재귀 함수로 찾은 최대공약수는:", result_recursive)
```

코딩새싹 🌱: `if b == 0:` 이 부분이 "나머지가 0이면 멈춰!" 하는 약속이네요! 그렇지 않으면 `gcd1(b, a % b)`로 자기 자신을 또 부르고요!

선생님: 정확해! 재귀 함수는 이렇게 "언제 멈출지(base case)"를 정해주는 게 아주 중요하단다. 그렇지 않으면 함수가 영원히 자기 자신을 부르다가 지쳐버릴 테니까!

궁금이 💡: 반복문으로 만든 gcd 함수랑 재귀 함수로 만든 gcd1 함수랑 결과는 똑같이 나오는데, 어떤 게 더 좋은 거예요?

선생님: 둘 다 유클리드 호제법의 원리를 잘 표현하는 훌륭한 방법이야! 때로는 반복문이 이해하기 쉽고, 때로는 재귀 함수가 문제의 구조를 더 우아하게 표현하기도 한단다. 지금은 두 가지 방법 모두 우리가 최대공약수라는 '숫자들의 숨은 대장'을 찾는 멋진 도구라는 것을 아는 것이 중요해!

선생님: (두루마리를 닫으며) 자, 오늘 우리는 아주 오랜 옛날부터 내려온 유클리드 아저씨의 지혜를 파이썬 코드로 직접 구현해봤어. 복잡해 보이는 문제도 이렇게 간단한 규칙을 반복해서 해결할 수 있다는 것, 정말 놀랍지 않니? 이것이 바로 알고리즘의 힘이란다! 오늘 탐험도 정말 멋졌어! 🏛️☒👍