<[파이썬 알고리즘 탐험] 제1장. 비밀의 숫자, 소수를 찾아라!>

선생님: (숫자들이 적힌 카드 중에서 2, 3, 5, 7, 11 같은 카드들을 보여주며) 얘들아, 안녕! 이 숫자들의 공통점이 뭘까? 바로 약수가 1과 자기 자신밖에 없는 아주 특별하고 외로운(?) 친구들, **소수**란다!

코딩새싹 🌱: 아! 1보다 큰 자연수 중에서 1과 자기 자신으로만 나누어떨어지는 수요! 2, 3, 5, 7… 맞죠? 그럼 4나 6은 약수가 더 있으니까 소수가 아니고요!

선생님: 정확해! 4(1,2,4)나 6(1,2,3,6)처럼 약수가 3개 이상인 친구들은 **합성수**라고 부르지. 그럼 우리가 어떤 숫자를 딱 봤을 때, 이 숫자가 소수인지 합성수인지 어떻게 알 수 있을까?

궁금이 💡: 음… 2부터 자기 자신보다 작은 숫자들로 다 나눠보면 되지 않을까요? 그래서 한 번이라도 나누어떨어지면 합성수고, 끝까지 안 나누어떨어지면 소수요!

선생님: 아주 좋은 생각이야! 그 아이디어를 코드로 한번 옮겨볼까?

<1. 첫 번째 시도: 나누기 특공대! (하지만 약간의 함정이?)>

```
number = int(input("자연수 하나를 입력하세요: "))

# 2부터 number - 1 까지의 숫자로 나눠보자!
for i in range(2, number):
  if number % i == 0: # 만약 나누어떨어지는 i를 찾았다면,
    print(f"{number}는 합성수입니다.") # 그 즉시 합성수라고 외치고,
    break # 더 이상 확인할 필요 없으니 반복문 탈출!

print("소수입니다.") # 어? 이 줄은 언제 실행될까?
```

코딩새싹 🌱: break를 만나면 for문을 빠져나오니까… 만약 6을 입력하면 i가 2일 때 "6는 합성수입니다."가 나오고 끝나겠네요!

궁금이 💡: (고개를 갸웃하며) 선생님, 그런데 만약 6을 입력해서 "합성수입니다."가 출력되고 break로 빠져나오면, 그 다음에 있는 print("소수입니다.")도 실행되는 거 아니에요? 그럼 6은 합성수이면서 소수가 되는 이상한 상황이…?!

선생님: (빙긋 웃으며) 이야, 궁금이가 아주 예리하게 함정을 발견했구나! 맞아. 지금 이 코드는 break를 만나든 만나지 않든, for문이 끝나면 무조건 print("소수입니다.")를 실행하게 되어 있어. 6 같은 합성수는

“합성수입니다.”와 “소수입니다.” 둘 다 외치게 되고, 7 같은 소수는 “소수입니다.”만 외치게 되겠지. 뭔가 어설프지?

코딩새싹 🌱: 네! 하나의 숫자가 합성수이면서 동시에 소수일 수는 없잖아요!

선생님: 그래서 이런 판단은 “이 숫자는 소수인가? (True/False)”라는 질문에 명확하게 답해주는 **함수**로 만드는 게 훨씬 깔끔하단다!

<2. 업그레이드! ‘소수 판별 탐정’ 함수 만들기!>

선생님: 자, `is_prime(숫자)`라는 이름의 함수를 만들어서, 이 함수가 소수이면 `True`를, 아니면 `False`를 돌려주도록 설계해보자!

```
def is_prime(num):
  # 1단계: 소수의 기본 조건! 2보다 작은 숫자는 소수가 아니야.
  if num < 2:
    return False # 바로 False 돌려주고 탐정 임무 종료!

  # 2단계: 2부터 num-1까지 나눠보기 (궁금이의 아이디어!)
  # 그런데 더 똑똑한 방법이 있어! 바로 숫자의 "제곱근까지만" 나눠보는 거야!
  # 왜냐하면, 만약 num의 약수 중에 제곱근보다 큰 수가 있다면,
  # 반드시 그 짝꿍이 되는 제곱근보다 작은 약수가 이미 존재하기 때문이지!
  # 예를 들어 36의 약수는 1,2,3,4,6,9,12,18,36 인데, 제곱근 6까지만 확인해도 돼!
  # (num ** 0.5) 가 제곱근을 구하는 마법! int()로 정수 부분만 가져오고, +1을 해줘서 그 범위까지 포함!
  for i in range(2, int(num ** 0.5) + 1):
    if num % i == 0: # 나누어떨어지는 약수를 찾았다면,
      return False # 더 볼 것도 없이 이 숫자는 소수가 아니야! 바로 False!

  # 3단계: 위대한 탐정의 결론!
  # for 반복문을 무사히 다 통과했다는 건? 중간에 약수를 하나도 못 찾았다는 뜻!
  return True # 그러니까 이 숫자는 소수다! True!
```

궁금이 💡: 와! `int(num ** 0.5) + 1` 이 부분 정말 똑똑한데요! 숫자가 엄청 커도 제곱근까지만 확인하면 되니까 훨씬 빠르게 판단할 수 있겠어요!

코딩새싹 🌱: 그리고 return False를 만나면 함수가 바로 끝나버리니까, 아까처럼 "합성수입니다."랑 "소수입니다."가 같이 나올 걱정도 없겠네요!

선생님: 정확해! 이 is_prime 함수만 있으면 이제 어떤 숫자든 소수인지 아닌지 확실하게 판별할 수 있단다!

```
number = int(input("자연수 하나를 입력하세요: "))
if is_prime(number): # is_prime 함수의 결과가 True라면,
  print(f"{number}는 소수입니다.")
else: # False라면,
  print(f"{number}는 소수가 아닙니다. (합성수이거나 1 이하의 수)")
```

<3. 소수 탐정단, 대출동! (2부터 1000까지 소수 찾기)>

선생님: 자, 그럼 우리가 만든 강력한 is_prime 함수를 이용해서, 2부터 1000까지의 자연수 중에서 소수인 친구들만 쏙쏙 골라내서 리스트로 만들어볼까?

```
primes = [] # 소수를 담을 빈 리스트 준비!

for i in range(2, 1001): # 2부터 1000까지의 숫자를 하나씩 꺼내서,
  if is_prime(i): # 만약 is_prime 탐정이 "이 숫자 소수 맞아요!(True)" 라고 하면,
    primes.append(i) # 소수 리스트에 추가!

print("\n--- 2부터 1000까지의 소수들 ---")
print(primes)
```

코딩새싹 🌱: 와! is_prime() 함수를 만드니까 for문 안의 코드가 정말 간단해졌어요! 함수는 정말 마법 같아요!

선생님: (흐뭇하게) 그렇지? 오늘 우리는 소수라는 특별한 숫자를 찾는 알고리즘을 고민해보고, 처음의 어설픈 시도에서 문제점을 발견하고, 더 똑똑하고 효율적인 is_prime 함수로 발전시키는 과정을 경험했어. 이렇게 문제를 해결하는 논리적인 방법을 찾고, 더 좋은 방법으로 개선해나가는 것이 바로 알고리즘 탐험의 즐거움이란다! 오늘 정말 멋진 탐정이었어, 우리 꼬마 알고리즈머들! 🕵☒⇳☒