<[파이썬 숫자 탐험 고급편] 제1장: 분수의 세계와 한 줄 마법, 람다!>

선생님: (케이크 조각과 자를 보여주며) 얘들아, 안녕! 우리가 1 나누기 3을 하면 0.3333… 이렇게 끝없이 이어지는 소수가 나오지? 가끔은 이렇게 소수점으로 딱 떨어지지 않는 숫자를 '분수' 그 자체로 정확하게 다루고 싶을 때가 있단다. 그럴 때 파이썬은 우리에게 Fraction이라는 아주 정교한 도구를 선물해 줘!

<1. 정확한 계산의 세계! fractions 모듈>

```
from fractions import Fraction # '분수' 마법 도구를 가져오자!

# 분수 만들기! Fraction(분자, 분모)
a = Fraction(2, 4) # 2/4 라는 분수를 만들면...
print(f"분수 a는? {a}") # 약분까지 해서 1/2 로 똑똑하게 보여주네!

b = Fraction(3, 5) # 3/5
print(f"분수 b는? {b}")
```

코딩새싹 🌱: 와! Fraction(2, 4)라고 했는데, 알아서 1/2로 약분까지 해줘요! 정말 똑똑한데요!

선생님: 그렇지? 이 Fraction 객체는 분자와 분모를 따로따로 가지고 있단다.

```
print(f"분수 a의 분자: {a.numerator}") # .numerator 로 분자를!
print(f"분수 a의 분모: {a.denominator}") # .denominator 로 분모를!
```

선생님: 그리고 이 분수끼리 더하거나 빼거나, 심지어 제곱하는 것도 아주 정확하게 계산해준단다!

```
print(f"\n{a} + {b} = {a + b}") # 분수 + 분수 = 분수!
print(f"{a} - {b} = {a - b}") # 통분도 알아서 척척!

num_fraction = Fraction(1, 4) ** 2 # (1/4)의 제곱은? 1/16!
print(f"(1/4)의 제곱은 분수로: {num_fraction}")
```

궁금이 💡: 그럼 이 분수를 우리가 쓰는 소수로 바꾸고 싶으면 어떻게 해요?

선생님: 좋은 질문이야! float() 변신 마법을 쓰면 된단다. 그리고 그렇게 바꾼 소수에는 넘파이의 수학 도구도 쓸 수 있지!

```
# 분수를 소수로 변신!
float_from_fraction = float(num_fraction) # 1/16을 소수로!
print(f"분수 {num_fraction}을 소수로 바꾸면: {float_from_fraction}") # 0.0625

import numpy as np # 넘파이 마법사 소환!
print(f"소수 {float_from_fraction}의 제곱근은?: {np.sqrt(float_from_fraction)}") # 0.25

# Fraction 객체에 바로 소수점 제곱을 해도 결과는 float!
num2 = Fraction(2, 3)
```

```
print(f"분수 {num2}의 0.5 제곱 (루트)은?: {num2 ** 0.5}") # 결과는 float
```

<2. 분수 문제 자판기! RationalNumber 클래스 만들기>

선생님: 자, 이제 이 Fraction 마법을 이용해서, 우리만의 '유리수(분수) 문제 자판기' 클래스를 만들어볼까?

```
class RationalNumber:
  def __init__(self):
    # 분자를 만들 숫자 4개를 -10에서 9 사이에서 무작위로 뽑자!
    # self.constant[0]/self.constant[1]  +  self.constant[2]/self.constant[3]
    # 주의: 분모가 0이 되면 안되니, 실제로는 0을 제외하는 로직이 필요하지만 여기선 일단 생략!
    self.constants = np.random.randint(-9, 10, 4) # 분모가 0이 안되게 -9부터! (0이 아닌 정수 뽑는 방법은
    # 만약 분모로 뽑힌 숫자가 0이면, 0이 아닌 다른 숫자로 바꿔주는 처리를 해주는 게 좋아!
    if self.constants[1] == 0: self.constants[1] = 1 # 임시로 1로!
    if self.constants[3] == 0: self.constants[3] = 1 # 임시로 1로!

    self.operation_type = np.random.randint(1) # 어떤 연산을 할지? 0~0 사이에서 무작위 선택! (앗! 이것밖
    print("✨ 유리수 문제 자판기 준비 완료! ✨")

  def give_question(self): # 이름을 give_question으로 바꿔봤어!
    print("\n--- 오늘의 분수 문제! ---")
    # 현재 operation_type이 항상 0이니까, 덧셈 문제만 나오겠네!
    if self.operation_type == 0:
      print(f"{self.constants[0]}/{self.constants[1]} + {self.constants[2]}/{self.constants[3]} = ?")
    # 여기에 elif self.operation_type == 1: (뺄셈) 등을 추가하면 더 다양한 문제를 낼 수 있겠지?
    # 이 메소드는 화면에 출력만 하니까, return 값은 없어 (None)

  def tell_answer(self): # 이름도 tell_answer로!
    print("--- 정답 공개! ---")
    if self.operation_type == 0:
      frac1 = Fraction(self.constants[0], self.constants[1])
      frac2 = Fraction(self.constants[2], self.constants[3])
      return frac1 + frac2 # Fraction 객체를 결과로 돌려준다!
    # 다른 연산들도 여기에 추가할 수 있겠지?
```

코딩새싹 🌱: self.constants에 숫자 4개를 한 번에 뽑는 거 신기해요! np.random.randint(-9, 10, 4) 이렇게요!

궁금이 💡: (코드를 유심히 보며) 선생님! self.operation_type = np.random.randint(1) 이라고 되어있는데, randint(1)은 0밖에 못 만들지 않아요? 그럼 이 자판기는 항상 덧셈 문제만 내는 거 아니에요?

선생님: (눈을 찡긋하며) 오오, 역시 궁금이 탐정! 매의 눈으로 핵심을 정확히 짚었구나! 맞아. 지금 이 코드대로라면 우리 분수 자판기는 '덧셈 전문가'가 되어버렸네! 만약 np.random.randint(2)로 바꾸면 0 또는

1이 나오니까 뺄셈 문제도 추가할 수 있겠지? 이렇게 작은 부분을 바꿔서 기능을 확장하는 게 프로그래밍의 재미란다!

코딩새싹 🌱: 아까 코드에 `print(problem.question())` 이런 부분이 있었는데, 제가 그렇게 해보니까 문제 나오고 다음 줄에 `None`이라고 떠요! 왜 그렇죠?

선생님: (고개를 끄덕이며) 아주 중요한 발견이야! `give_question` 메소드는 문제를 화면에 `print`하기만 하고, 특별히 `return`하는 값이 없지? 함수나 메소드가 명시적으로 `return`하는 값이 없으면, 파이썬은 '돌려줄 게 없네~'라는 뜻으로 `None`을 돌려준단다. 그래서 `print(problem.give_question())`은 `give_question`이 화면에 문제를 출력한 후, 그 `None`이라는 반환값을 다시 `print`하게 되는 거지!

```
# 자판기 사용해보기!
problem = RationalNumber()
problem.give_question() # 이 메소드는 화면에 출력만 하고 끝! (반환값은 None)
print(problem.tell_answer()) # 이 메소드는 Fraction 객체를 반환하므로, 그 값이 출력됨!
```

<3. 한 줄 마법! 람다(lambda) 함수>

선생님: 우리가 `def`를 써서 여러 줄로 함수를 만들었지? 그런데 아주아주 간단한 기능을 하는 함수는 딱 한 줄로도 만들 수 있는 마법이 있단다. 그게 바로 **람다(lambda) 함수**야! 이름 없는 '익명 함수'라고도 부르지.

`lambda 재료들 : 재료들을_이용한_결과식`

```
# x를 받아서 x의 제곱을 돌려주는 간단한 함수
square_spell = lambda x: x ** 2
print(f"\n람다 마법으로 3의 제곱은? {square_spell(3)}")

# x를 받아서 짝수인지 아닌지(True/False) 알려주는 함수
is_even_spell = lambda x: x % 2 == 0
print(f"람다 마법으로 7은 짝수인가? {is_even_spell(7)}")
print(f"람다 마법으로 4는 짝수인가? {is_even_spell(4)}")
```

코딩새싹 🌱: 와! `def`랑 `return` 없이도 함수가 만들어졌어요! 정말 한 줄 마법 같은데요! 언제 쓰면 좋아요?

선생님: 람다 함수는 이렇게 기능이 아주 간단하거나, 다른 함수의 재료로 잠깐 쓰이고 말 함수를 만들 때 유용하단다. 복잡한 함수는 여전히 `def`로 만드는 게 좋지!

<4. 아직 정의되지 않은 마법? (prime_fraction)>

선생님: 아까 코드 조각 중에 `prime_fraction(num)` 이라는 것이 있었는데, 우리가 이 주문은 아직 배우거나 만들지 않았지? 아마도 분모를 소인수분해하는 새로운 마법 주문을 만들려고 했던 것 같은데, 그건 우리 다음 모험에서 한번 도전해볼까? (미소)

<5. 넘파이의 작은 선물들 (np.sqrt, np.pi)>

선생님: 마지막으로, 넘파이는 복잡한 배열 계산뿐만 아니라, 이렇게 간단한 수학 상수(`np.pi` 원주율)나 함수(`np.sqrt()` 제곱근)도 우리에게 선물로 준단다!

```
print(f"\n넘파이가 알려주는 원주율(pi): {np.pi}")
print(f"넘파이로 계산한 4의 제곱근: {np.sqrt(4)}")
```

선생님: (다양한 분수 조각들을 보여주며) 자, 오늘 우리는 숫자의 세계를 더 깊이 탐험하며 정확한 분수를 다루는 `Fraction` 마법과, 간결함의 미학을 보여주는 `lambda` 한 줄 마법을 배웠어! 그리고 우리가 만든 클래스에 이 새로운 마법들을 적용해서 더 똑똑한 문제 자판기도 만들 수 있다는 가능성도 보았지. 파이썬의 세계는 정말 무궁무진하구나! 오늘 탐험도 정말 훌륭했어! 🌟