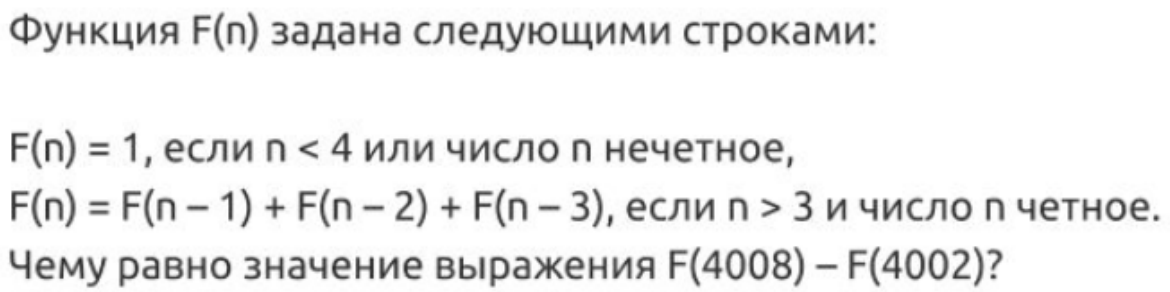
Задание №16, Рекурсия

Теория + Алгоритм решения

Рекурсия — это поведение функции, при котором она вызывает сама себя

План решения

аналог демо-варианта 2025:


Функция F(n) задана следующими строками:

F(n) = 1, если n < 4 или число n нечетное,
F(n) = F(n – 1) + F(n – 2) + F(n – 3), если n > 3 и число n четное.
Чему равно значение выражения F(4008) – F(4002)?

1. Объявляем функцию (*название может быть любое, кроме уже существующих в Python)*:

```
def F(n):
```

2. Пишем тело функции согласно условию *(не забываем табуляцию)*:

```
    if n < 4 or n % 2 != 0:
        return 1
    if n > 3 and n % 2 == 0:
        return F(n-1) + F(n-2) + F(n-3)
```

3. Выводим то, что нас просят найти:

```
print(F(4008) - F(4002))
```

4. Пробуем запустить. Если мы получаем ответ, то задача решена! Весь код:

```
def F(n):
  if n < 4 or n % 2 != 0:
     return 1
  if n > 3 and n % 2 == 0:
        return F(n-1) + F(n-2) + F(n-3)
print(F(4008) - F(4002))
```

5. Однако всё чаще на ЕГЭ встречаются усложнённые задания, в которых не так всё просто. В том числе и в этой задаче: если вы запустите этот код, получите ошибку “RecursionError: maximum recursion depth exceeded”, которая означает, что глубина рекурсии превысила установленный по умолчанию лимит в 1000. Поэтому важно знать ещё 4 способа решения:
   a. При ошибке лимита рекурсии (как выше), просто увеличиваем этот лимит, добавляя в самом начале 2 строчки:

      ```
      import sys
      sys.setrecursionlimit(10000) # Не советую больше, может зависнуть П
      ```

   b. Если код слишком долго считается, то используем кэширование (сохраняем результаты для каждого посчитанного F(n), чтобы переиспользовать их, а не считать заново):

      ```
      from functools import lru_cache

      @lru_cache() # Обязательно ставим прямо над объявлением функции
      def F(n):
      ```

      Иногда, как в этом случае, нужно предварительно вычислить значения функции F(n) для некоторого количества n, чтобы при вызове F(4008) и F(4002) функция уже имела все необходимые значения в кэше:

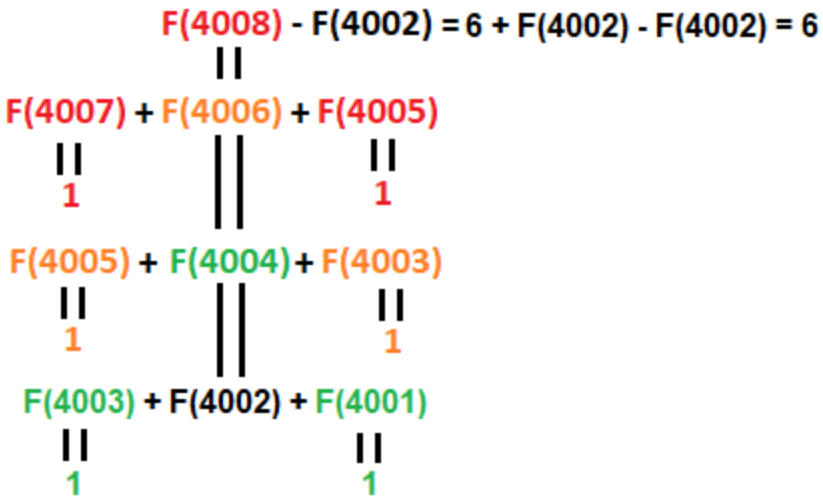
      ```
      for n in range(1,4000):
            F(n) # Предварительно считаем значения функции для n < 4000
      ```

```
print(F(4008) - F(4002))
```

** Можно реализовать и самописное кэширование через список или словарь:*

```
results = [-1] * 4300 # -1 ⇒ значение не подсчитано
def F(n):
    if results[n] != -1:
        return results[n]
    if n < 4 or n % 2 != 0:
        return 1
    if n > 3 and n % 2 == 0:
        return F(n-1) + F(n-2) + F(n-3)
for i in range(1,4020):
    results[i] = F(i) # Считаем результаты)
print(results[4008] - results[4002])
```

c. Часто, если просят найти, чему равняется выражение от нескольких вызовов функций для разных n, как в этом случае, можно преобразовать члены этого выражения в соответствии с функцией и сократить:



d. Также можно решить задачу динамически. Если вручную подставить некоторые числа и посмотреть на результат функции, можно понять, что она делает, и написать аналогичную функцию без рекурсии. В данном случае, **F(n) = 1** для нечетных **n** и **F(n) = n-1** для четных:

```
for n in range(1, 10):
    print(f'{n}: {F(n)}')
# Вывод:
# 1: 1
# 2: 1
# 3: 1
# 4: 3
# 5: 1
# 6: 5
# 7: 1
# 8: 7
# 9: 1
# 10: 9
```

Напишем аналогичную функцию, не используя рекурсию, и найдем ответ:

```
def F_new(n):
    if n % 2 != 0:
        return 1
    else:
        return n -1
print(F_new(4008) - F_new(4002)) # Вывод: 6
```

💡 Кэширование и лимит рекурсии можно использовать вместе, но формально оба способа - “читерство”, т.к. это задание с каждым годом всё более нацелено на решение руками или динамически. Задачу, в решении которой не поможет ли расширение лимита, ни кэширование, придумать сложно, но тем не менее можно и такая вполне может быть на экзамене.