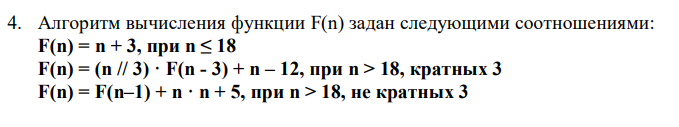
Л.р. №4 по ФЛП

Чередов Р.А., гр. ПИ-02

Задание № 4



import sys  
import timeit  
from functools import lru\_cache  
from pympler import asizeof  
# Рекурсивная функция  
def F(n):  
 if n <= 18:  
 return n+3  
 elif n % 3 == 0:  
 return (n//3) \* F(n-3)+n-12  
 else:  
 return F(n-1)+n\*n+5  
start = timeit.default\_timer()  
print('Обычная функция:')  
print(F(990))  
end = timeit.default\_timer()  
print('Время:', end - start)  
print('Память:', asizeof.asizeof(F))  
print('--------------------------------------------------------------------------------------------')  
# Мемоизация  
#при мемоизации, функция сохраняет значения. Если результат функции уже сохранен  
#то функция возвращает его вместо того чтобы вычислять снова  
  
  
  
memo = {0: 0, 1: 1} #создаем словарь для сохр-я результатов функции, словарь играет роль кэша  
def F\_memo(n):  
 if n not in memo:  
 if n % 3 == 0:  
 memo[n] = (n//3) \* F\_memo(n-3)+n-12  
 else:  
 memo[n] = F\_memo(n-1)+n\*n+5  
 return memo[n]  
start = timeit.default\_timer()  
print('Мемоизация:')  
print(F\_memo(990))  
end = timeit.default\_timer()  
print('Время:', end - start)  
print('Память:', asizeof.asizeof(memo))  
print('--------------------------------------------------------------------------------------------')  
# Реализация с декоратором lru\_cache  
#Декоратор @lru\_cache является способом реализации мемоизации, но с некоторыми отличиями от обычной мемоизации.  
#основное отличие в том, что декоратор более гибкий и удобный  
#он автоматически управляет кэшем и применяет стратегию "Least Recently Used" (наименее недавно использованный)  
# для удаления редко используемых результатов из кэша, если он достигает максимального размера.  
  
@lru\_cache(maxsize=None)  
def F\_cache(n):  
 if n <= 18:  
 return n+3  
 elif n % 3 == 0:  
 return (n//3) \* F\_cache(n-3)+n-12  
 else:  
 return F\_cache(n-1)+n\*n+5  
  
  
  
start = timeit.default\_timer()  
print('Рализация с декоратором')  
print(F\_cache(990))  
end = timeit.default\_timer()  
print('Время:', end - start)  
print('Память:', asizeof.asizeof(F\_cache))  
class FIterator:  
 def \_\_init\_\_(self, limit):  
 self.limit = limit  
 self.count = 0  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if self.count > self.limit:  
 raise StopIteration  
 value = F(self.count)  
 self.count += 1  
 return value  
  
  
# Проверка итератора  
start = timeit.default\_timer()  
print('Итератор:')  
iterator = FIterator(5)  
for value in iterator:  
 print(value,"-->", end="")  
  
end = timeit.default\_timer()  
print('\nВремя:', end - start)  
print('Память:', asizeof.asizeof(F\_cache))  
print('\n--------------------------------------------------------------------------------------------')  
  
def FGenerator(n):  
 count = 0  
 while count <= n:  
 yield F(count)  
 count += 1  
  
# Проверка генератора  
end = timeit.default\_timer()  
print('Генератор:')  
generator = FGenerator(5)  
for value in generator:  
 print(value,"-->",end="")  
  
end = timeit.default\_timer()  
print('\nВремя:', end - start)  
print('Память:', asizeof.asizeof(F\_cache))

