


Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего
образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

Лабораторная работа No 6
Итераторы, генераторы и декораторы

Вариант 9-2-4-1.

Автор: Киселёва Анна Николаевна

подпись 

Факультет: ФБИТ

Группа: N3147

Преподаватели:

Грозов Владимир Андреевич



Санкт-Петербург 2025

Задание:

9 - Модифицированный генератор Фибоначчи

2 - Способ реализации ГПСЧ (Генератор (Функция, содержащая оператор yield))

4 - @shuffle_str_args

Декоратор должен заменить любые строки, которые передаются как позиционные или именованные аргументы в функцию или метод, на строки, в которых символы исходных строк переставлены в случайном (с помощью ГПСЧ) порядке.

1 - Способ реализации декоратора (Функция)

Алгоритм ГПСЧ:

@shuffle_str_args - Декоратор должен заменить любые строки, которые передаются как позиционные или именованные аргументы в функцию или метод, на строки, в которых символы исходных строк переставлены в случайном (с помощью ГПСЧ) порядке.

Генератор реализован на основе динамического вычисления последовательности Фибоначчи с запаздыванием. Где числа k , j называют запаздыванием.

```
def fib_gen(j: int = 24, k: int = 55, mod: int = 2**32) -> Iterator[int]:
    a = [i for i in range(55)]
    i = k
    while True:
        num = (a[i - j] + a[i - k]) % mod
        if i < len(a):
            a[i] = num
        else:
            a.append(num)
        i += 1
        yield num
```

Значения $j = 24$, $k = 55$, $\text{mod} = 2^{32}$ были взяты из описания Аддитивного ГПСЧ из источника Генераторы псевдослучайных чисел, Слеповичев, 2017.

Примеры работы программы: Декорирование функции

Пример работы с позиционным аргументом

```
>>> from prg6ankN3147_rnd import MyPRNG, fib_gen
>>> from prg6ankN3147_subst import TypeError, shuffle_str_args
>>> gen = fib_gen()
>>> prng = MyPRNG(gen)
>>> @shuffle_str_args(prng, file_name="shuffle_log.txt")
... def example_function(string: str) -> str:
...     return string
...
>>> example_function("Alice")
'Acile'
```

Вывод в файл shuffle_log.txt

```
anna@Huaweianna:~/projects/prg6ankN3147$ tail shuffle_log.txt
04.06.25 13:20:53 example_function(args=('Alice',), kwargs={}) -> Acile
04.06.25 13:20:53 example_function: аргумент Alice (args[0]) заменен на 'Acile'
anna@Huaweianna:~/projects/prg6ankN3147$
```

Пример такой же, но отсутствует вывод в файл, логи выводятся в консоль

```
>>> from prg6ankN3147_rnd import MyPRNG, fib_gen
>>> from prg6ankN3147_subst import TypeError, shuffle_str_args
>>> gen = fib_gen()
>>> prng = MyPRNG(gen)
>>> @shuffle_str_args(prng)
... def example_function(string: str) -> str:
...     return string
...
>>> example_function("Alice")
04.06.25 13:26:31 example_function(args=('Alice',), kwargs={}) -> Acile
04.06.25 13:26:31 example_function: аргумент Alice (args[0]) заменен на 'Acile'

'Acile'
```

Декорирование всего класса:

Пример работы с именованными аргументами

```
anna@Huaweianna:~/projects/prg6ankN3147$ python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from prg6ankN3147_rnd import MyPRNG, fib_gen
>>> from prg6ankN3147_subst import TypeError, shuffle_str_args
>>> gen = fib_gen()
>>> prng = MyPRNG(gen)
>>> @shuffle_str_args(prng, file_name="shuffle_log.txt")
... class Boo:
...     def __init__(self, x: str):
...         self.x = x
...     def get_x(self) -> str:
...         return self.x
...     def set_x(self, y : str) -> str:
...         return self.x + y
...
>>> boo = Boo('Bottle of ')
>>> boo.set_x(y='Juice')
'Bottle of Jciue'
```

Вывод в файл:

```
04.06.25 13:34:44 set_x(args=(), kwargs={'y': 'Juice'}) -> Bottle of Jciue
04.06.25 13:34:44 set_x: аргумент Juice (kwargs['y']) заменен на 'Jciue'
anna@Huaweianna:~/projects/prg6ankN3147$
```

Декорирование метода класса:

```
anna@Huaweianna:~/projects/prg6ankN3147$ python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from prg6ankN3147_rnd import MyPRNG, fib_gen
m prg6ankN3147_subst import TypeError, shuffle_s>>> from prg6ankN3147_subst import
TypeError, shuffle_str_args
b_gen()
prng = MyPRNG(gen)
class Boo:
    def __init__(self, x: str):
        self.x = x
    def get_x(self) -> str:
        return self.x
    @shuffle_str_args>>> gen = fib_gen()
>>> prng = MyPRNG(gen)
>>> class Boo:
...     def __init__(self, x: str):
...         self.x = x
...     def get_x(self) -> str:
...         return self.x
...     @shuffle_str_args(prng, file_name="shuffle_log.txt")
...     def set_x(self, y: str) -> str:
...         return self.x + y
...     def sort_x(self, y: str):
...         s = ''
...         for i in sorted(y):
...             s += str(i)
...         return s
...
>>> boo = Boo('Bottle of ')
```

Метод `set_x` должен вернуть перемешанное слово Water тк используется декоратор

Метод `sort_x` должен вернуть отсортированную в алфавитном порядке строку, не перемешанную, тк нет декоратора

```
>>> boo = Boo('Bottle of ')
>>> boo.set_x('Water')
'Bottle of Wetar'
>>> boo.sort_x(y = 'avbavbavb')
'aaabbbvvv'
>>> |
```

Вывод в файл `shuffle_log.txt` изменений

```
04.06.25 13:53:22 set_x(args=('Water',), kwargs={}) -> Bottle of Wetar
04.06.25 13:53:22 set_x: аргумент Water (args[0]) заменен на 'Wetar'
anna@Huaweianna:~/projects/prg6ankN3147$ |
```

По заданию декоратор должен перемешивать строки
пример с вводом чисел, вызывающий ошибку TypeError

```
anna@Huaweianna:~/projects/prg6ankN3147$ python3
Python 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from prg6ankN3147_rnd import MyPRNG, fib_gen
>>> from prg6ankN3147_subst import TypeError, shuffle_str_args
>>> gen = fib_gen()
>>> prng = MyPRNG(gen)
>>> @shuffle_str_args(prng, file_name="shuffle_log.txt")
... def test_func(a: str, b: str):
...     return a + b
...
>>> test_func(123, 321)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/home/anna/projects/prg6ankN3147/prg6ankN3147_subst.py", line 50, in wrappe
r
    raise TypeError(f"Аргумент {arg} не является строкой")
prg6ankN3147_subst.TypeError: Аргумент 321 не является строкой
>>> |
```

Проверка модуля Pytest

```
anna@Huaweianna:~/projects/prg6ankN3147$ pytest
===== test session starts =====
platform linux -- Python 3.10.12, pytest-8.3.5, pluggy-1.6.0
rootdir: /home/anna/projects/prg6ankN3147
collected 8 items

test_prg6ankN3147_rnd.py .... [ 50%]
test_prg6ankN3147_subst.py .... [100%]

===== 8 passed in 0.02s =====
anna@Huaweianna:~/projects/prg6ankN3147$ |
```

Исходный текст программы: test_prg6ankN3147_rnd.py

```
1  ✓ from prg6ankN3147_rnd import MyPRNG, fib_gen
2      import pytest
3      @pytest.fixture() 10 usages
4  ✓ def prng():
5      gen = fib_gen()
6      return MyPRNG(gen)
7
8  ▶ ✓ def test_shuffled_string(prng):
9      """Проверка работы метода класса по перемешиванию строк с помощью ГПСЧ"""
10     shuffle_str = ['abcdefg', '12345667', 'barabulka']
11     ✓ for i in shuffle_str:
12         shuffle = prng.shuffle_str(i)
13         assert shuffle != i
14     assert prng.shuffle_str('aaa') == 'aaa'
15     assert prng.shuffle_str('') == ''
16
17  ▶ ✓ def test_next_int(prng):
18     """проверка метода вывода следующего целого числа"""
19     res = prng.next_int()
20     assert isinstance(res, int)
21
22  ▶ ✓ def test_next_float(prng):
23     """проверка метода вывода следующего вещественного числа"""
24     res = prng.next_float()
25     assert isinstance(res, float)
26
27  ▶ ✓ def test_next_str(prng):
28     """Получает строки src и возвращает строки, содержащую символы из src, переставленные в случайном порядке"""
29     s = set()
30     ✓ for i in range(10):
31         res = prng.next_str()
32         assert len(res) <= 100
33         assert isinstance(res, str)
34         s.add(res)
35     assert len(s) == 10
```

prg6ankN3147_rnd.py

```
from abc import ABC, abstractmethod
from collections.abc import Iterator

#генератор псевдослучайных чисел на основе последовательности Фибоначчи с запаздыванием
def fib_gen(j: int = 24, k: int = 55, mod: int = 2**32) -> Iterator[int]: 5 usages
    a = [i for i in range(55)]
    i = k
    while True:
        num = (a[i - j] + a[i - k]) % mod
        if i < len(a):
            a[i] = num
        else:
            a.append(num)
        i += 1
        yield num

class PRNGBase(ABC): 1 usage
    @abstractmethod
    def next_int(self) -> int:
        pass

    @abstractmethod
    def next_float(self) -> float:
        pass

    @abstractmethod
    def next_str(self) -> str:
        pass

    @abstractmethod
    def shuffle_str(self, src: str) -> str:
        pass
```



```

class MyPRNG(PRNGBase): 6 usages
35 def __init__(self, gen: Iterator):
36     self.gen = iter(gen)
37
38 def next_int(self) -> int: 1 usage
39     """Возвращает случайное целое число"""
40     return( next(self.gen) )
41
42 def next_float(self) -> float: 1 usage
43     """Возвращает случайное вещественное число в диапазоне от 0.0 до 1.0 (не включая)"""
44     return( next(self.gen) / 2**32 )
45
46 def next_str(self) -> str: 1 usage
47     """Возвращает случайную последовательность символов"""
48     # получаемся Число из ГПСЧ по модулю 256 - код символа ascii
49     max_len = 100
50     string = []
51     len_str = next(self.gen) % max_len #ограничение длины генерированной строки
52     for i in range(len_str):
53         num_ascii = next(self.gen) % 256
54         string.append(chr(num_ascii))
55     return(''.join(string))
56
57 def shuffle_str(self, src: str) -> str: 5 usages
58     """Получает строку src и возвращает строку, содержащую символы из src, переставленные в случайном порядке"""
59     string = list(src)
60     for i in range(len(string)):
61         j = next(self.gen) % (i+1) #символ исходной строки от 0 до i
62         t = string[i] #меняем символы местами
63         string[i] = string[j]
64         string[j] = t
65     return(''.join(string))

```

prg6ankN3147_subst.py

```

from prg6ankN3147_rnd import MyPRNG, fib_gen
from datetime import datetime
from functools import wraps

class TypeError(Exception): 4 usages
    pass

def shuffle_str_args(prng: MyPRNG, file_name: str = None): 5 usages
    def decorator(func):

        if isinstance(func, type): # если функция это на самом деле класс, то нужно применить декоратор ко всем методам
            for name, method in func.__dict__.items(): #смотрим все методы класса
                if callable(method) and name != "__init__": #пропускаем инициализацию класса
                    setattr(func, name, decorator(method)) #применяем wrapper ко всем остальным методам
            return func #возвращаем функцию, которая на самом деле класс, для которого были продекорированы все методы кроме __init__

        else: # если func не класс а функция
            @wraps(func)
            def wrapper(*args, **kwargs): #позиционные и именованные аргументы функции

                #если декоратор вызывается для отдельного метода класса, то args = [self, ...]
                #значит нам надо убрать self обработать остальные аргументы и вернуть self обратно для итога вызова func

                self = None
                if not isinstance(args[0], str):
                    self = args[0] #запоминаем self
                    args = args[1:] #обрезаем args

                start_time = datetime.now().strftime("%d.%m.%Y %H:%M:%S")

                logs = [] #изменения
                shuffled_args = [] #для позиционных
                shuffled_kwargs = {} #для именованных

```

```
result = func(*shuffled_args, **shuffled_kwargs)
# формирование полного лога |
full_log = log_start + f" -> {result}"
if logs:
    full_log += "\n" + "\n".join(logs)

if file_name: #запись логов в указанный файл
    with open(file_name , 'a') as file:
        file.write(full_log + '\n')
else: #если имя файла не указано делаем логи в консоль
    print(full_log + '\n')

return result
return wrapper
return decorator
```

test_prg6ankN3147_subst.py

```
1 from prg6ankN3147_rnd import MyPRNG, fib_gen
2 from prg6ankN3147_subst import TypeError, shuffle_str_args
3 import pytest
4
5 @pytest.fixture() # фикстура для создания экземпляра класса 8 usages
6 def prng():
7     gen = fib_gen()
8     return MyPRNG(gen)
9
10 def test_dec_function(prng, tmp_path):
11     """тест для проверки использования декоратора для функции"""
12     file_path = tmp_path / "func_log.txt"
13
14     @shuffle_str_args(prng, file_name=str(file_path))
15     def test_func(password: str, username: str):
16         return f"user: {username}, password: {password}"
17
18     res = test_func(password: '1234567', username: 'Anna')
19     #полученные с помощью ГПСЧ результат != исходному
20     assert res != 'user: Anna, password: 1234567'
21
22     assert file_path.exists()
23     assert file_path.stat().st_size > 0
24
```

```
4 def test_dec_class_method(prng, tmp_path):
5     file_path = tmp_path / "class_log.txt"
6     """проверка использования декоратора для всего класса"""
7     @shuffle_str_args(prng, file_name=str(file_path))
8     class Boo:
9         def __init__(self, x: str):
10             self.x = x
11
12         def get_x(self) -> str:
13             return self.x
14
15         def set_x(self, y: str) -> str:
16             return self.x + y
17
18         def sort_x(self, y: str):
19             s = ''
20             for i in sorted(y):
21                 s+=str(i)
22             return s
23
24     boo = Boo('Bottle of ')
25     res = boo.set_x(y='Water') # возвращаем перемешанное слово Water
26     res_2 = boo.sort_x(y = 'abcdefg') #возвращает отсортированную строку, тк декорируется аргумент
27     assert res != 'Bottle of Water'
28     assert res_2 == 'abcdefg'
29     assert file_path.exists()
30     assert file_path.stat().st_size > 0
31     assert boo.get_x() == 'Bottle of '
32
```

```

> def test_dec_class(prng, tmp_path):
    file_path = tmp_path / "class_method_log.txt"
    """проверка использования декоратора для метода класса"""
    class Boo:
        def __init__(self, x: str):
            self.x = x

        def get_x(self) -> str:
            return self.x

        @shuffle_str_args(prng, file_name=str(file_path))
        def set_x(self, y: str) -> str:
            return self.x + y

        def sort_x(self, y: str):
            s = ''
            for i in sorted(y):
                s+=str(i)
            return s

    boo = Boo('bottle of ')
    res_1 = boo.set_x('Water')
    res_2 = boo.sort_x(y = 'abcdefg')
    assert res_1 != 'Bottle_of_Water'
    assert res_2 != 'gfedcba'

    assert file_path.exists()
    assert file_path.stat().st_size > 0

```

```

84
85 > def test_invalid_integer(prng, tmp_path):
86     """тест на попытку передачи не строки, вызов TypeError"""
87     file_path = tmp_path / "func_log.txt"
88
89     @shuffle_str_args(prng, file_name=str(file_path))
90 > def test_func(a: str, b: str):
91     return a + b
92
93     res = test_func(a: "123", b: "321")
94     assert isinstance(res, str)
95
96     with pytest.raises(TypeError):
97         test_func(a: 123, b: 321)
98
99     assert file_path.exists()

```