Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем и технологий»

**Отчёт**

***по Лабораторной работе №23***

по дисциплине ***«Языки программирования»***

Исполнитель: Колосов Артём Александрович

Группа: 10701219

Преподаватель: Лапанович Ирина Олеговна

2020

Цель работы:

Изучить и освоить рекурсивный способ решения сложных задач; научиться гра-

мотно проектировать и декомпозировать (разбивать) задачи на более мелкие

фрагменты с использованием итерационных и рекурсивных алгоритмов и

практически закрепить полученные знания на примере разработки интерак-

тивных приложений.

Основное задание:

Для решения каждой задачи нужно реализовать минимум две реализации:

одна реализация на базе циклов, а другая – на базе рекурсии.

1) Требуется описать функцию, которая бы определяла, является ли заданное

число точной степенью двойки (или тройки и т.д.).

2) Опишите функцию, которая вычисляет сумму цифр числа.

3) Опишите функцию, которая вычисляет сумму элементов заданной одномерной

последовательности.

4) Опишите функцию, которая реверсирует элементы заданной одномерной по-

следовательности.

5) Необходимо описать функцию *power(x, n)*, вычисляющую xn для любого веще-

ственного x(≠0) и любого целого n. Предусмотреть два рекурсивных алгоритма

реализации: простой алгоритм нахождения степени числа и ускоренный. Оце-

нить с использованием Big O алгоритмическую сложность данных вариантов.

6) Требуется описать функцию *f(x, n)*, вычисляющую величину xn/n! при любом ве-

щественном *x* и любом неотрицательном целом *n*.

7) Необходимо реализовать функцию, которая вычисляет N-элемент ряда Фибо-

наччи. На базе данной функции разработать программу, которая должна пред-

лагать пользователю следующие возможности: вывод конкретного элемента

последовательности; вывод всех элементов последовательности до указанного

пользователем элемента; или той части последовательности, значение послед-

него элемента которой не превосходит введённое пользователем значение).

Требования к выполнению:

1) Необходимо спроектировать блок-схемы алгоритмов решений соответствую-

щих заданий согласно своему варианту (варианты назначаются преподавате-

лем или выбираются самостоятельно студентом, но ни один вариант не дол-

жен повторяться у студентов из одной группы) и на базе данных алгоритмов

разработать интерактивные консольные приложения с использованием архи-

тектурного шаблона проектирования ***MVC***.

2) Для каждого алгоритма решения задания оценить (вычислить) алгоритмиче-

скую сложность по необходимому времени и затрачиваемой памяти исполь-

зуя определение ***Big O***.

3) Для масштабируемости разрабатываемого программного решения размер

последовательности (списка) и его элементы должны задаваться пользовате-

лем во время выполнения программы или с помощью генератора псевдослу-

чайных чисел.

4) Для автоматизации заполнения различными значениями искомого контей-

нера рекомендуется использовать соответствующие функции генерирования

псевдослучайных чисел модуля ***random***.

5) Рекомендуется избегать использования глобальных переменных при написа-

нии основной логики приложения.

6) Если логически не подразумевается или в заданиях иного не указано, то вход-

ными и выходными данными являются вещественные числа (числа с плаваю-

щей запятой).

7) Все программы должны быть разбиты на отдельные функции. Среди данных

функций рекомендуется добавлять стартовую функцию ***main***, с которой лучше

производить запуск программы.

8) При выполнении заданий необходимо по максимуму пытаться разрабатывать

универсальный, масштабируемый, легко поддерживаемый и читаемый код.

9) Также рекомендуется придерживаться ***Single Responsibility Principle****,* ***SRP***

(принципа единственной ответственности) – постарайтесь вынести основную

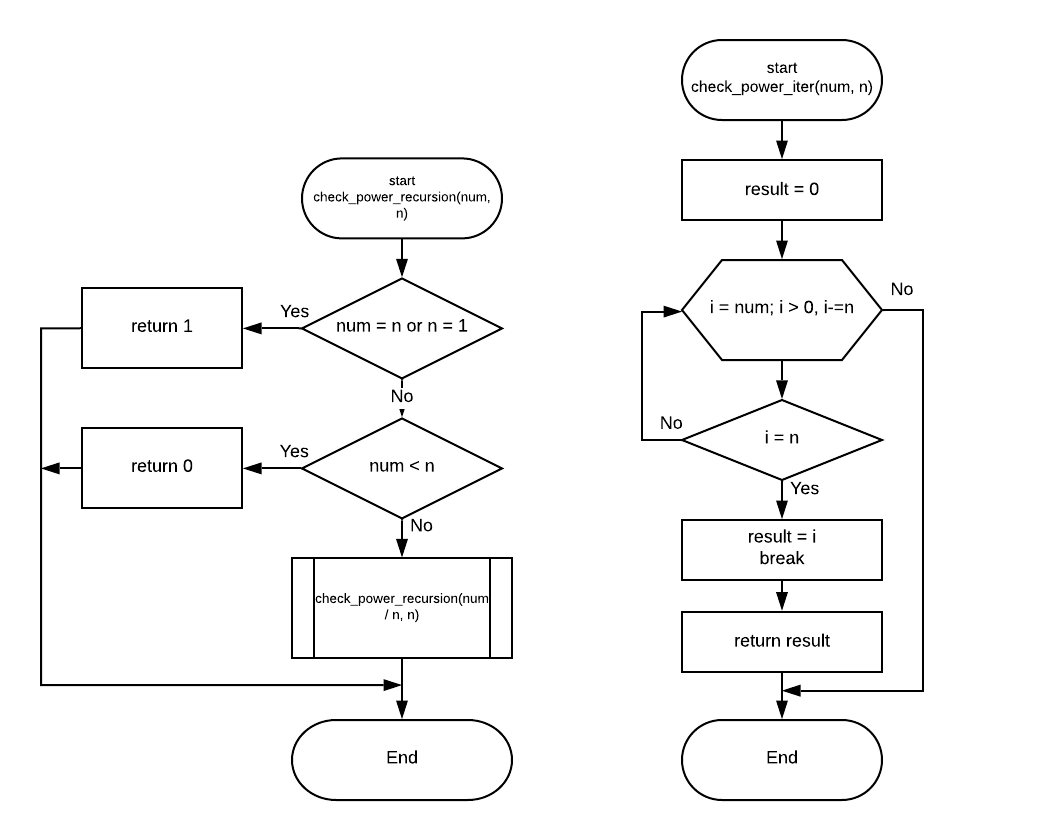
бизнес-логику задания в отдельную функцию (т.е. архитектура приложения

должна минимум состоять из нескольких функций).

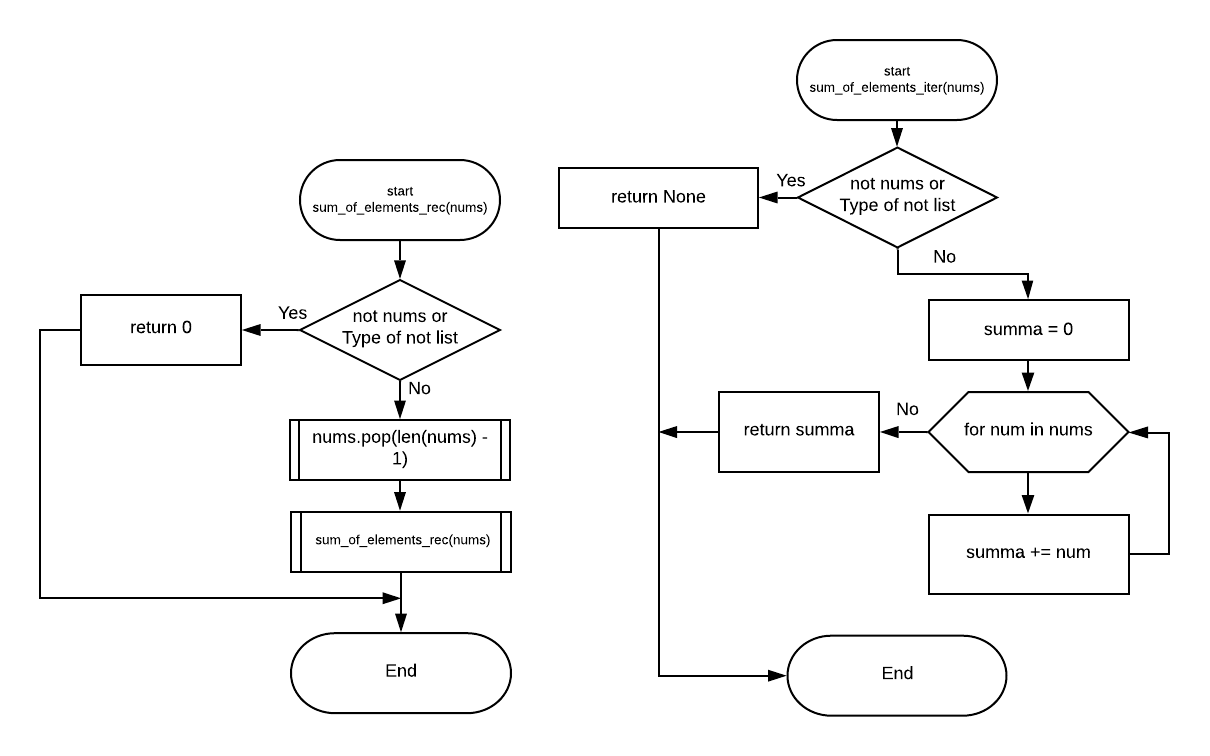
10) В соответствующих компонентах (функциях) бизнес-логики необходимо

предусмотреть «защиту от дурака».

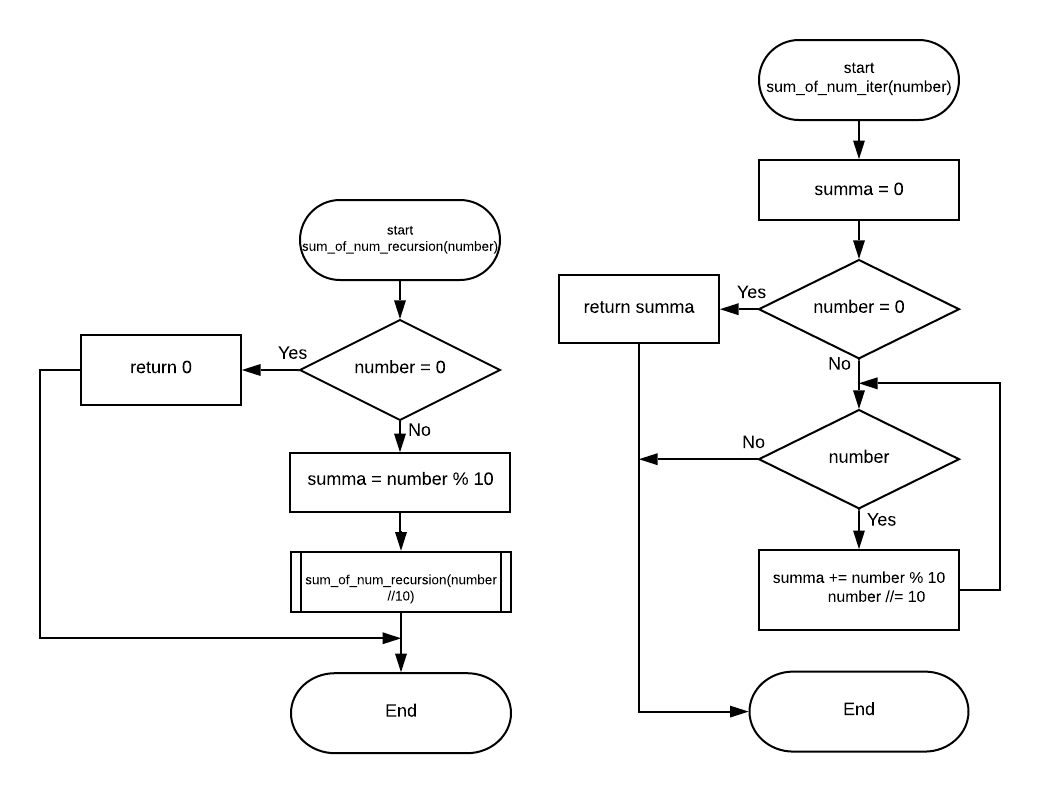
Блок-схемы основных алгоритмов:



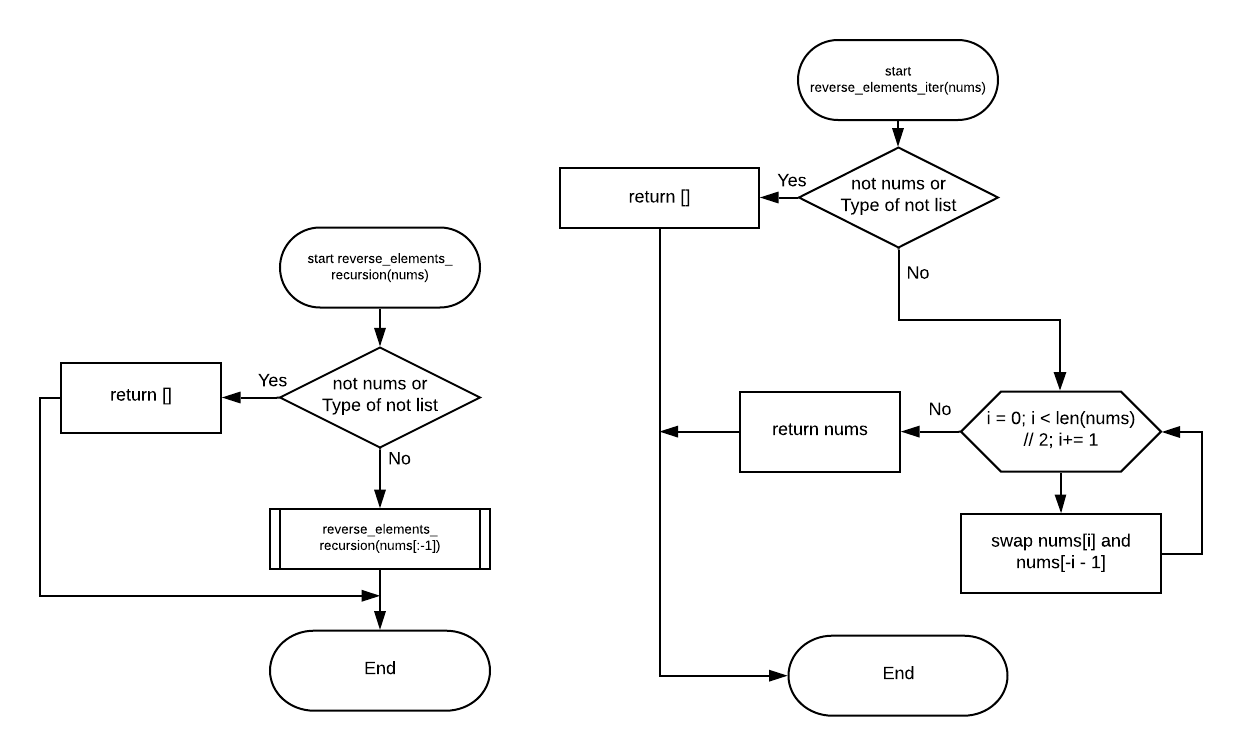
**Рис 23.1 Блок-схемы функций для проверки степени числа (реализации через рекурсию и циклы)**

****

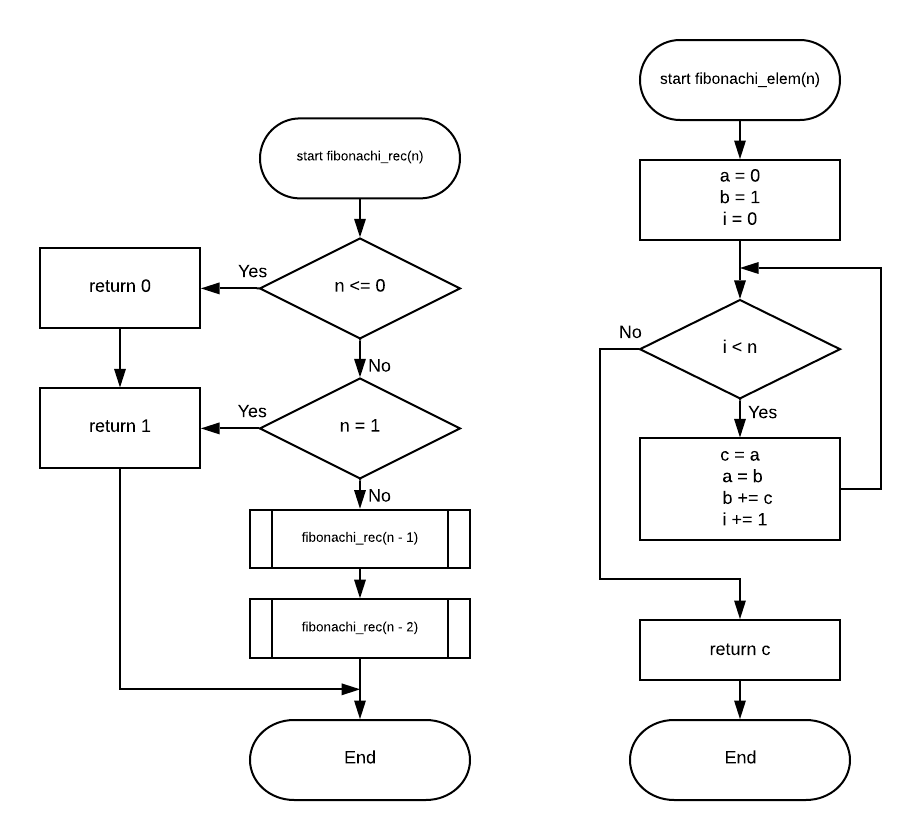
**Рис 23.2 Блок-схемы функций для поиска суммы в последовательности (реализации через рекурсию и циклы)**



**Рис 23.3 Блок-схемы функций для поиска суммы всех чисел из другого числа (реализации через рекурсию и циклы)**

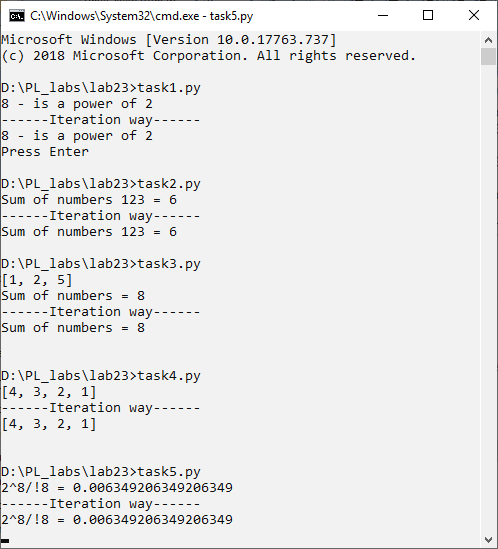


**Рис 23.4 Блок-схемы функций для реверсирования последовательности (реализации через рекурсию и циклы)**

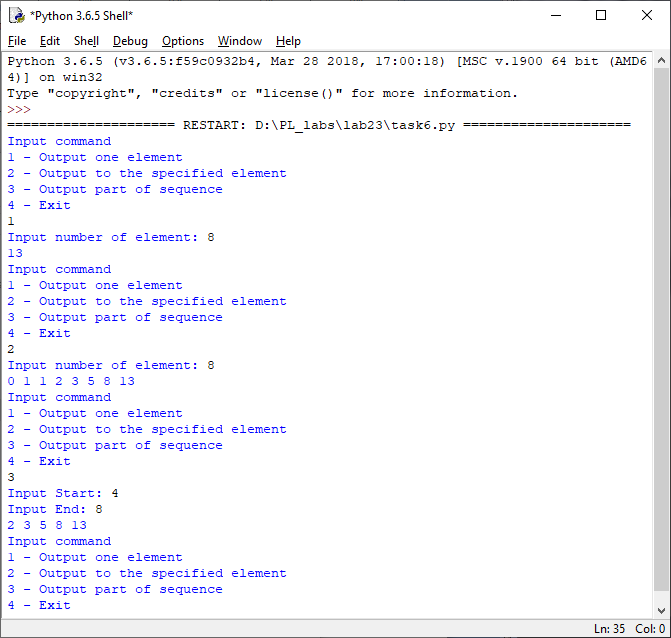


**Рис 23.5 Блок-схемы функций для поиска чисел фибоначи (реализации через рекурсию и циклы)**

Результаты выполнения основного задания:



**Рис 23.6 Результаты выполнения основных заданий №1-5**



**Рис 23.7 Результат выполнения основного задания №6**

Оценка сложности некоторых алгоритмов:

Числа фибоначи через рекурсию:

Сложность: O(2^N)

Память: O(N)

Числа фибоначи через цикл:

Сложность: O(N)

Память: O(1)

Проверка степени через рекурсию:

Сложность: O(logn(num))

Память: O(N)

Что я освоил в процессе выполнения лабораторной работы (выводы):

1. Рекурсивные и итерационные алгоритмы различаются по подходам проектирования программы, скорости выполнения, и использованию памяти
2. Рекурсия полезный инструмент, но который стоит использовать только тогда, когда это даст большую пользу
3. Некоторые задачи легче реализовать при помощи рекурсии

*ПРИЛОЖЕНИЕ A*

Листинг исходных кодов программ

**Файл task1.py**

# Программа проверяет возможность сущ. степени

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def check\_power\_recursion(num, n):

if num == n or n == 1:

return 1

if num < n:

return 0

return check\_power\_recursion(num / n, n)

def check\_power\_iter(num, n):

result = 0

for i in range(num, 0, -n):

if i == n:

result = 1

break

return result

num = 8

n = 2

if check\_power\_recursion(num, n):

print("{0} - is a power of {1}".format(num, n))

print("Iteration way".center(25, "-"))

if check\_power\_iter(num, n):

print("{0} - is a power of {1}".format(num, n))

input("Press Enter")

**Файл task2.py**

# Программа находит сумму всех чисел

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def sum\_of\_num\_recursion(number):

if number == 0:

return 0

summa = number % 10

return summa + sum\_of\_num\_recursion(number // 10)

def sum\_of\_num\_iter(number):

if number == 0:

return 0

summa = 0

while number:

summa += number % 10

number //= 10

return summa

n = 123

print("Sum of numbers {0} = {1}".format(n, sum\_of\_num\_recursion(n)))

print("Iteration way".center(25, "-"))

print("Sum of numbers {0} = {1}".format(n, sum\_of\_num\_iter(n)))

**Файл task3.py**

# Программа внаходи сумму элементов списка

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def sum\_of\_elements\_rec(nums):

if not isinstance(nums, list) or not nums:

return 0

return nums.pop(len(nums) - 1) + sum\_of\_elements\_rec(nums)

def sum\_of\_elements\_iter(nums):

if not isinstance(nums, list) or not nums:

return None

summa = 0

for num in nums:

summa += num

return summa

n = [1,2,5]

print(n)

print("Sum of numbers = {0}".format(sum\_of\_elements\_rec(n[:])))

print("Iteration way".center(25, "-"))

print("Sum of numbers = {0}".format(sum\_of\_elements\_iter(n)))

input()

**Файл task4.py**

# Программа разворачивает список

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def reverse\_elements\_recursion(nums):

if not isinstance(nums, list) or not nums:

return []

return [nums[-1]] + reverse\_elements\_recursion(nums[:-1])

def reverse\_elements\_iter(nums):

if not isinstance(nums, list) or not nums:

return []

for i in range(len(nums) // 2):

nums[i], nums[-i - 1] = nums[-i - 1], nums[i]

return nums

n = [1, 2, 3, 4]

print(reverse\_elements\_recursion(n))

print("Iteration way".center(25, "-"))

print(reverse\_elements\_iter(n))

input()

**Файл task5.py**

# Программа вычесляет x^n/n!

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def fact\_recursion(n=0):

if n == 0:

return 1

return n \* fact\_recursion(n - 1)

def fact\_iter(n):

fact = 1

for i in range(n, 0, -1):

fact \*= i

return fact

def pow\_rec(x, n):

if n == 0:

return 1

return x \* pow\_rec(x, n - 1)

def pow\_iter(x, n):

num = x

for i in range(1, n):

x \*= num

return x

def pow\_fact\_rec(x, n):

return pow\_rec(x, n) / fact\_recursion(n)

def pow\_fact\_iter(x, n):

return pow\_iter(x, n) / fact\_iter(n)

x = 2

n = 8

print("{0}^{1}/!{1} = {2}".format(x, n, pow\_fact\_rec(x, n)))

print("Iteration way".center(25, "-"))

print("{0}^{1}/!{1} = {2}".format(x, n, pow\_fact\_iter(x, n)))

input()

**Файл task6.py**

# Программа вычесляет элементы фибоначи

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 16.3.2020

def get\_input(msg):

return input(msg)

def fibonachi\_rec(n, k=1):

if n == 1 or n == 2 or n == k:

return 1

return fibonachi\_rec(n - 1) + fibonachi\_rec(n - 2)

def fibonachi\_elem(n):

a = 0

b = 1

i = 0

while i < n:

c = a

a = b

b += c

i += 1

return c

def output\_to\_user(msg="", s="", e="\n"):

print(msg, sep=s, end=e)

def all\_fib(n, k=1):

for i in range(k, n + 1):

output\_to\_user(fibonachi\_elem(i), e=" ")

output\_to\_user()

def build\_to\_int(var):

if not isinstance(var, str) or not var.isdigit():

return None

return int(var)

def main():

while True:

task = get\_input("Input command\n"

"1 - Output one element\n"

"2 - Output to the specified element\n"

"3 - Output part of sequence\n"

"4 - Exit\n")

if task == "1":

n = build\_to\_int(get\_input("Input number of element: "))

if n:

all\_fib(n, n)

elif task == "2":

n = build\_to\_int(get\_input("Input number of element: "))

if n:

all\_fib(n)

elif task == "3":

k = build\_to\_int(get\_input("Input Start: "))

n = build\_to\_int(get\_input("Input End: "))

if n and k:

all\_fib(n, k)

elif task == "4":

break

if \_\_name\_\_ == "\_\_main\_\_":

main()