Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по Практической работе №1**

по дисциплине: «Динамическое программирование»

Тема: «Изучение среды разработки, типов данных, операторов и функций вывода и ввода данных языка Python 3.\* на примере решения задач»

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС – 18.05  Федотов Ярослав |
| Проверила:  ассистент кафедры ПО  Корнеева Е.И. |

Тверь 2019

**«Наивный» алгоритм решения**

Алгоритм решения:

Ввод n

N1 = 0

N2 = 1

Функция fib0(принимаем n):

Если(n = 0) ,то

return 0

Всё\_если

Если(n = 1) ,то

return 1

Всё\_если

return fib0(n-1)+fib0(n-2)

Всё\_функция

Из time импортировать time

Start = time()

Пока (n2 < n)

N2 = fib0(n1)

N1 += 1

Печать n2

Всё\_пока

End = time()

Печать(Время: )

Если(n > 40),то

Печать(“После 40 наивный алгоритм работает медленее!”)

Печать(end-start)

Текст программы

print('Введите число, до которого мы будем двигаться')

n = int(input())

n1 = 0

n2 = 1

def fib0(n):

if(n == 0):

return 0

if (n == 1):

return 1

return fib0(n-1)+fib0(n-2)

from time import time

start = time()

while(n2 < n):

n2 = fib0(n1)

n1 += 1

print(n2)

end = time()

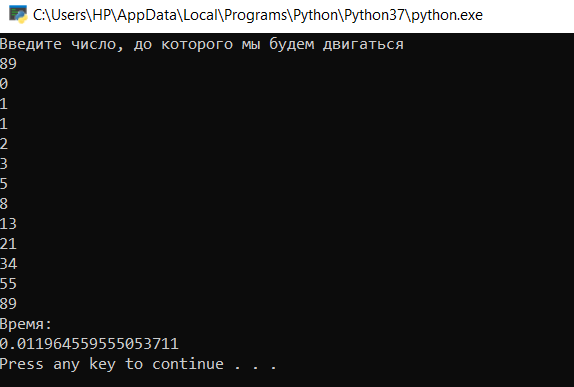
print('Время: ')

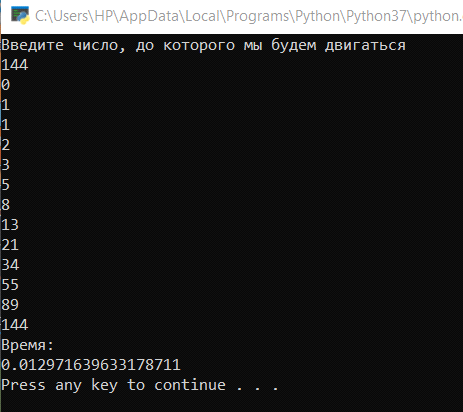
if(n > 40):

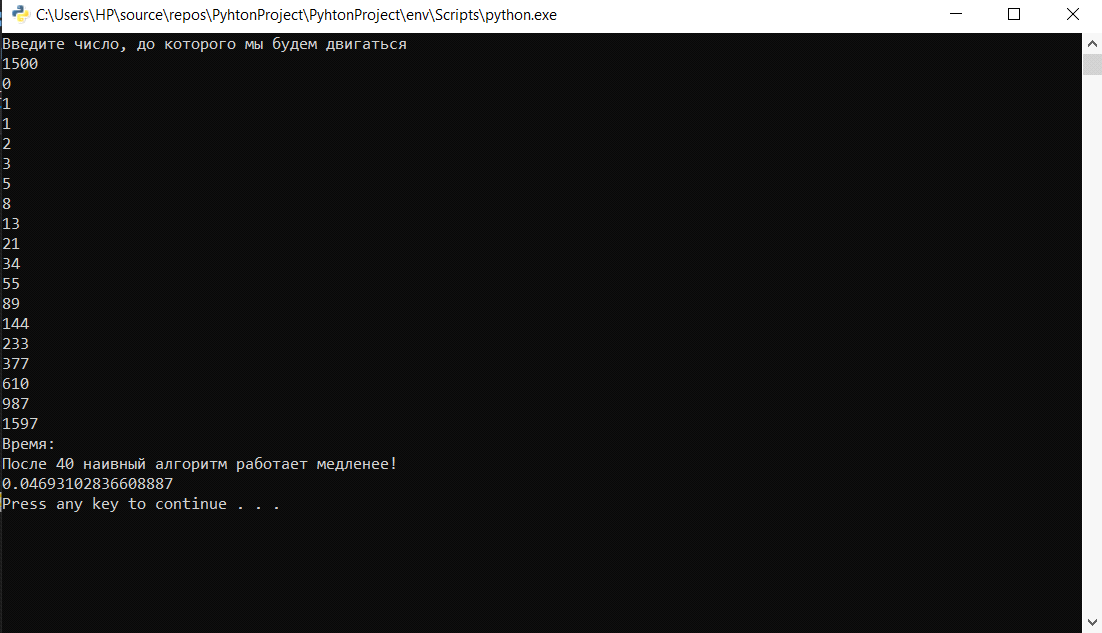
print('После 40 наивный алгоритм работает медленее!')

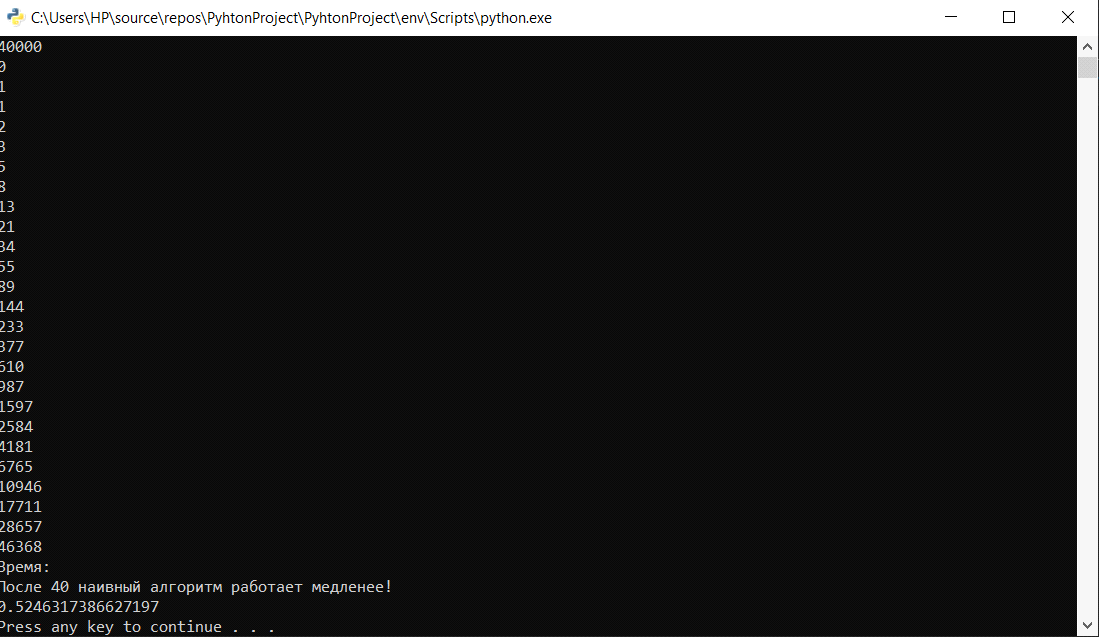
print(end - start)

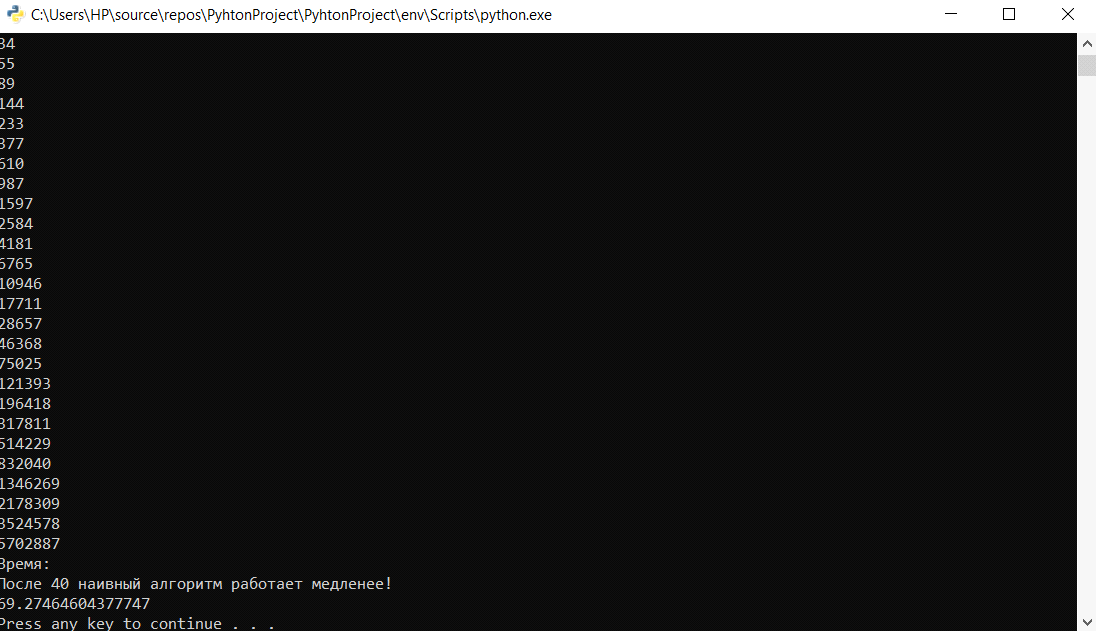
Скриншоты выполнения программы











Ответ на вопрос: Почему, по вашему мнению, алгоритм назван «наивным»?

Дело в том, что этот алгоритм является самым не эффективным, так как он перебирает все доступные значения при этом работает с теми значениями , с которыми он работал до этого, т.е как бы наивно верит в ,то что он допустил ,где-то ошибку и поэтому перепроверяет решение кучу раз!

**Рекурсивный алгоритм решения**

Алгоритм решения:

Ввод n

N1 = 0

N2 = 0

Функция fib(принимаем n):

Если n < 2 ,то

Return n

Всё\_если

Иначе

Return fib(n-1)+fib(n-2)

Всё\_иначе

Всё\_функция fib

Из time импортировать time

Start = time()

Пока (n2 < n)

N2 = fib(n1)

N1 += 1

Печать(n2)

End = time()

Всё\_пока

Печать(Время: )

Если(n > 40),то

Печать(“После 40 рекурсивный алгоритм работает медленее!”)

Печать(end – start)

Текст программы

print('Введите число, до которого мы будем двигаться')

n = int(input())

fib1 = fib2 = 1

from time import time

start = time()

while n > 0:

start = time()

fib1,fib2 = fib2, fib1+fib2

end = time()

n -= 1

print(fib2)

end = time()

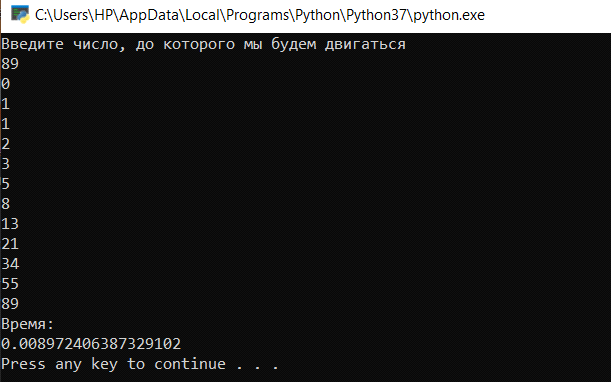
print('Время: ')

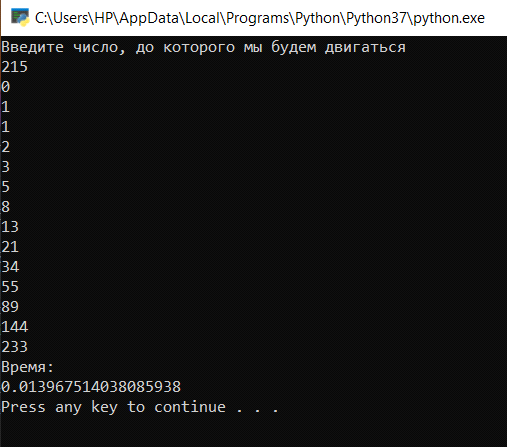
if(n > 40):

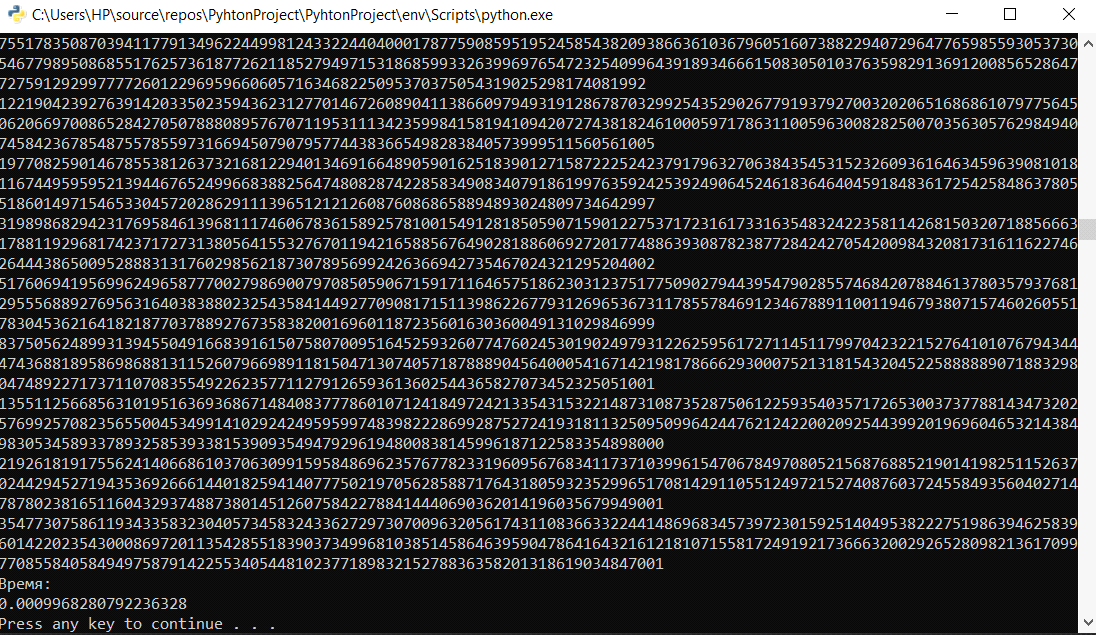
print('После 40 рекурсивный алгоритм работает медленее!')

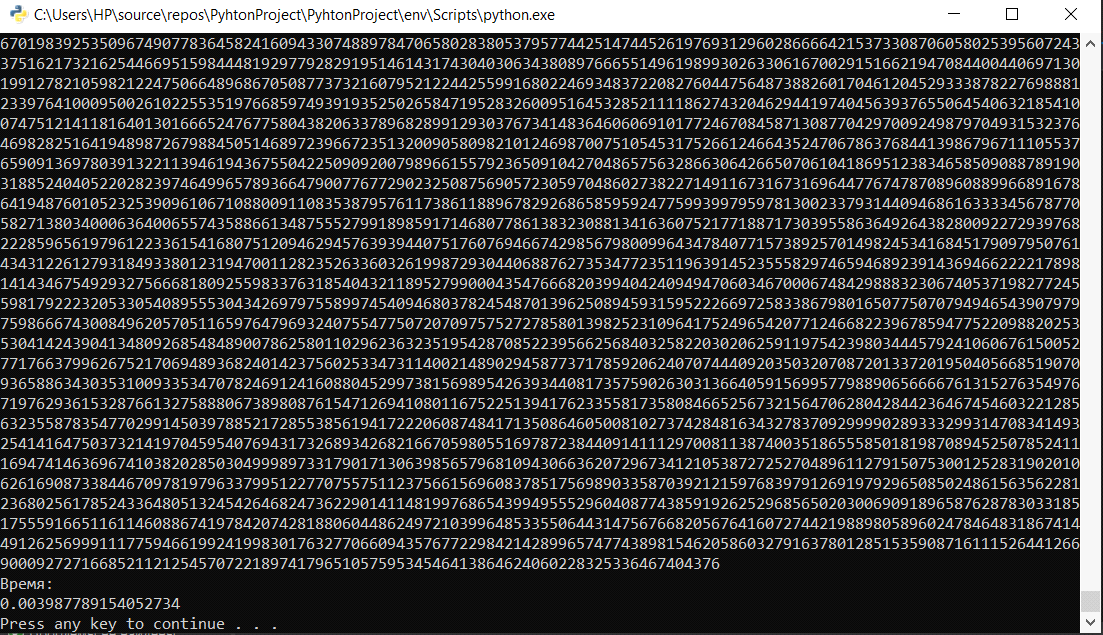
print(end - start)

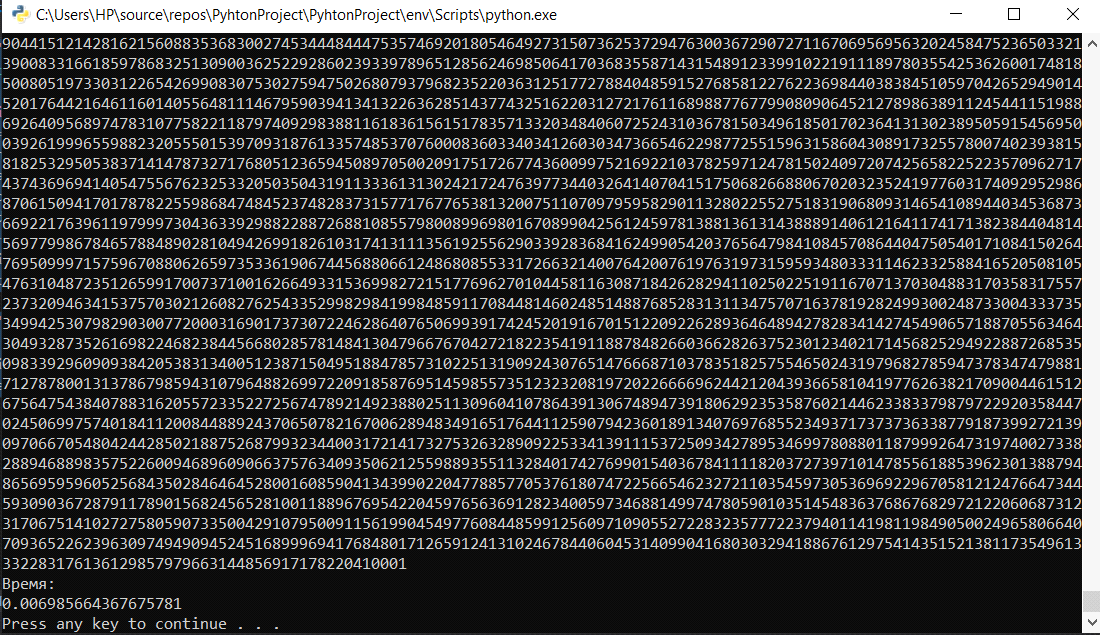
Скриншоты выполнения программы











**Решение с помощью алгоритма записи в массив**

Алгоритм решения:

Ввод n

Объявление списка F

Из time импортировать time

Start = time()

Добавить в список F 0

Добавить в список F 1

Для I от 2 до n

Добавить в список F[i-1]+F[i-2]

Всё\_для

Печать (F[len(F)-1])

End = time()

Печать(Время: )

Печать(end – start)

Текст программы

print('Введите число, до которого будет двигаться цикл for')

n = int(input())

F = []

from time import time

start = time()

F.append(0)

F.append(1)

for i in range(2,n):

F.append(F[i-1]+F[i-2])

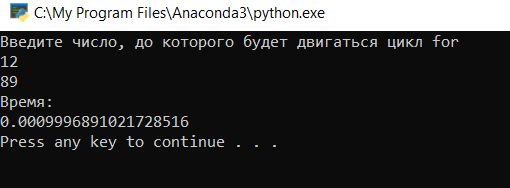
print(F[len(F)-1])

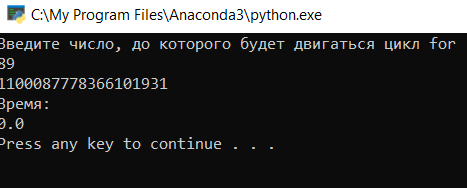
end = time()

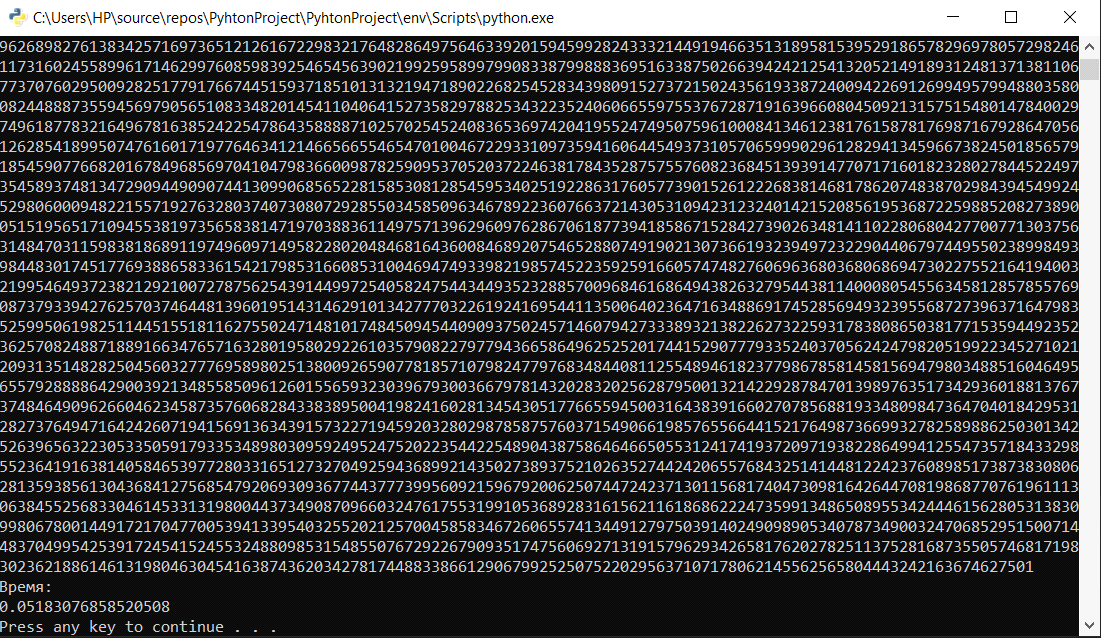
print('Время: ')

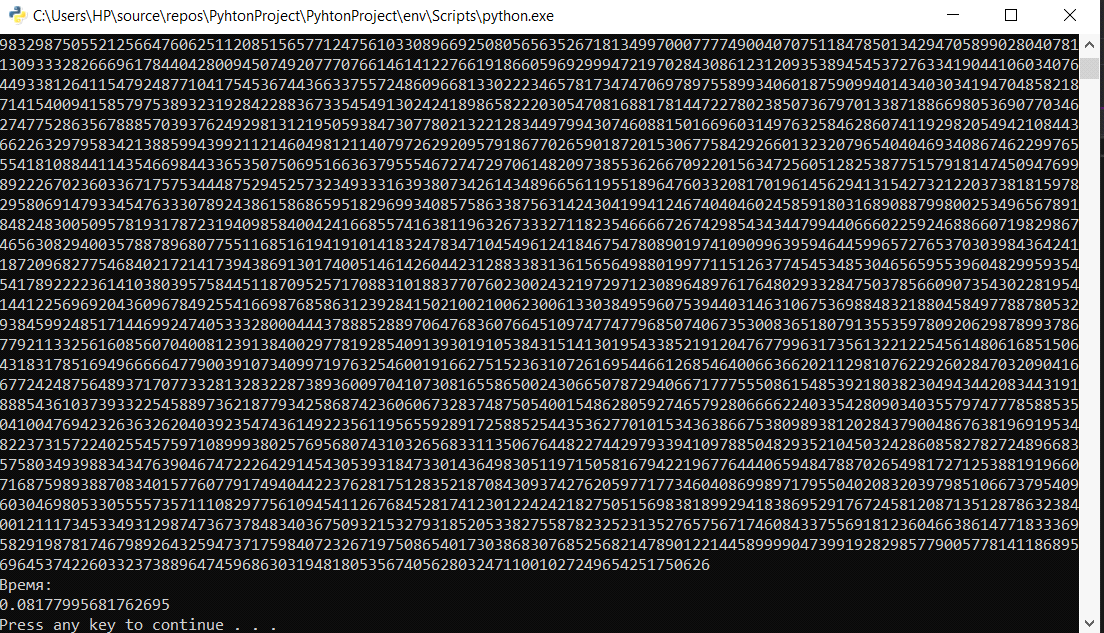
print(end - start)

Скриншоты выполнения программы









**Упрощённый алгоритм с записью в две переменные**

Алгоритм решения:

Ввод n

Из time импортировать time

Start = time()

N1 = 0

N2 = 0

Объявление целочисленных переменных I,a,b,c

A = 0

B = 1

Для I от 2 до n

С = a+b

A = b

B = c

Всё\_для

Печать(с)

End = time()

Печать(Время: )

Печать(end - start)

Текст программы

print('Введите число, до которого мы будем двигаться')

n = int(input())

from time import time

start = time()

n1 = 0

n2 = 1

i = int()

a = int()

b = int()

c = int()

a = 0

b = 1

for i in range(2,n):

c = a+b

a = b

b = c

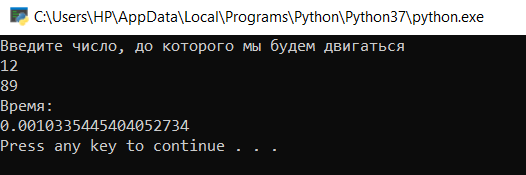
print(c)

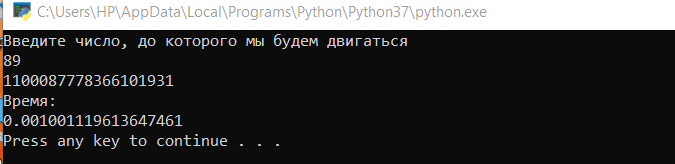
end = time()

print('Время: ')

print(end - start)

Скриншоты выполнения программы





**Метод вычисления чисел Фибоначчи с временной сложностью с записью соотношений в матричной форме**

Алгоритм решения:

Объявление класса MatrixFibonacci

Q = [[1,1],[1,0]]

Функция \_\_init\_\_(принимаем self)  
 из self берём длинное поле(МЕМО) = {}

Всё\_функция \_\_init\_\_

Функция MultiplyMatrices(принимаем self,M1,M2)

a11 = M1[0][0]\*M2[0][0] + M1[0][1]\*M2[1][0]

a12 = M1[0][0]\*M2[0][1] + M1[0][1]\*M2[1][1]

a21 = M1[1][0]\*M2[0][0] + M1[1][1]\*M2[1][0]

a22 = M1[1][0]\*M2[0][1] + M1[1][1]\*M2[1][1]

r = [[a11, a12], [a21, a22]]

return r

Всё\_функция MultiplyMatrices

Функция GetMatrixPower(принимаем self, M ,p)

Если (p = 1) , то

Return M

Если p в диапазоне self.\_\_memo,то

Return self.\_\_memo[p]

K = self.GetMatrixPower(M, int(p/2))

R = self.MultiplyMatrices(K, K)

self.\_\_memo[p] = R

return R

Всё\_функция GetMatrixPower

Функция GetNumber(self,n)

Если(n = 0) ,то

Return 0

Если(n = 1), то

Return 1

powers = [int(pow(2, b))

for (b, d) in enumerate(reversed(bin(n-1)[2:])) if d == '1']

matrices = [self.GetMatrixPower(MatrixFibonacci.Q, p)

for p in powers]

Пока (len(matrices) > 1)

M1 = matrices.pop()

M2 = matrices.pop()

R = self.MultiplyMatrices(M1,M2)

Добавить в список Matrices R

Return matrices[0][0][0]

Основной запуск:

Ввод n

Из time импортировать time

Start = time()

mfib = MatrixFibonacci()

Для i от 0 до n

num = mfib.GetNumber(i)

Всё\_для

Печать(num)

end = time()

Печать(Время: )

Печать (end - start)

Текст программы

class MatrixFibonacci:

Q = [[1, 1],

[1, 0]]

def \_\_init\_\_(self):

self.\_\_memo = {}

def MultiplyMatrices(self, M1, M2):

"""Умножение матриц

(ожидаются матрицы в виде списка список размером 2x2)."""

a11 = M1[0][0]\*M2[0][0] + M1[0][1]\*M2[1][0]

a12 = M1[0][0]\*M2[0][1] + M1[0][1]\*M2[1][1]

a21 = M1[1][0]\*M2[0][0] + M1[1][1]\*M2[1][0]

a22 = M1[1][0]\*M2[0][1] + M1[1][1]\*M2[1][1]

r = [[a11, a12], [a21, a22]]

return r

def GetMatrixPower(self, M, p):

"""Возведение матрицы в степень (ожидается p равная степени двойки)."""

if p == 1:

return M

if p in self.\_\_memo:

return self.\_\_memo[p]

K = self.GetMatrixPower(M, int(p/2))

R = self.MultiplyMatrices(K, K)

self.\_\_memo[p] = R

return R

def GetNumber(self, n):

"""Получение n-го числа Фибоначчи

(в качестве n ожидается неотрицательное целое число)."""

if n == 0:

return 0

if n == 1:

return 1

# Разложение переданной степени на степени, равные степени двойки,

# т.е. 62 = 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 32 + 16 + 8 + 4 + 1.

powers = [int(pow(2, b))

for (b, d) in enumerate(reversed(bin(n-1)[2:])) if d == '1']

matrices = [self.GetMatrixPower(MatrixFibonacci.Q, p)

for p in powers]

while len(matrices) > 1:

M1 = matrices.pop()

M2 = matrices.pop()

R = self.MultiplyMatrices(M1, M2)

matrices.append(R)

return matrices[0][0][0]

print('Введите число, до которого мы будем двигаться')

n = int(input())

from time import time

start = time()

mfib = MatrixFibonacci()

for i in range(0, n):

num = mfib.GetNumber(i)

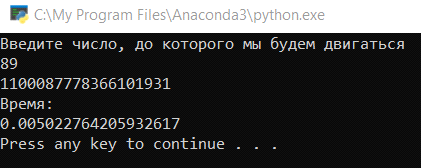
print(num)

end = time()

print('Время: ')

print(end - start)

Скриншоты выполнения программы



**Вычисление нужного числа по времени**

Для наивного алгоритма вычисляя числа Фибоначчи до 89 ушло времени:



До 144 ушло уже больше времени:



Для рекурсивного алгоритма вычисляя числа Фибоначчи до 89 ушло меньше времени чем у наивного алгоритма:



То же самое можно сказать и про работу проделанную алгоритмом до числа 233



Ещё быстрее двух предыдущих алгоритмов оказался алгоритм записи чисел Фибоначчи в массив, за счёт цикла for ,который на Python работает шустрее ,чем while и рекурсивные вызовы

До 89:



До 110087778366101931:



На равне с алгоритмом записи числа Фибоначчи в массив держится Упрощённый алгоритм с записью в две переменные:

До 89 приблизительно одинаковое время работы:

Различия лишь в тысячных долях, если их учесть, то упрощённый алгоритм с записью в две переменные работает быстрее чем алгоритм с записью в массив числа Фибоначчи.



До 110087778366101931:

Однако Упрощённый алгоритм с записью в две переменные показал себя на втором тесте медленнее, чем с записью чисел Фибоначчи в массив:



Алгоритм вычисления чисел Фибоначчи с временной сложностью с записью соотношений в матричной форме показал наивысшую скорость выполнения задачи:

До 89:



Вывод по работе:

Не смотря на то, что все алгоритмы, приведённые выше выполняют одну и туже задачу и скорость выполнения поставленной задачи у всех алгоритмов различается, так Наивный алгоритм оказался самым медленным, при выполнении задачи ,рекурсивный более быстрым, но не достаточно эффективным, алгоритм записи чисел Фибоначчи в массив занял третью строчку по эффективности выполнения задачи, вторую алгоритм вычисления чисел Фибоначчи с временной сложностью с записью соотношений в матричной форме и первую Упрощённый алгоритм с записью в две переменные так как матричный алгоритм записи не справился очень быстро с большим количеством чисел при n = 110087778366101931 его решение заняло около минуты.