Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №2**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Перевод исходной программы в обратную польскую запись»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Д. Воробьев

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

Краснодар

2024

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc103025612)

[2 Понятие обратной польской записи 3](#_Toc103025613)

[3 Алгоритм Дейкстры 4](#_Toc103025614)

[4 Перевод операторов цикла в ОПЗ 7](#_Toc103025615)

[5 Результаты экспериментов 9](#_Toc103025616)

[Приложение А Листинг программы и комментарии к нему 13](#_Toc103025617)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 2 | Python | Java |

Разработать программу для перевода закодированного текста исходной программы в обратную польскую запись.

Программа получает на входе файл – результат лексического анализа и строит обратную польскую запись исходной программы.

Отчет по работе должен содержать полное описание алгоритма Дейкстры: таблицу приоритетов операторов и операций, а также алгоритм работы со стеком. Листинг программы и комментарии к нему, пример.

**2 Понятие обратной польской записи**

Обратная польская запись (ОПЗ) – представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является расположение аргументов (операндов) перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи,

(a+d)/c+b\*(e+d),

в ОПЗ имеет следующее представление:

ad+c/bed+\*+.

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу ОПЗ может быть вычислена за один просмотр цепочки слева направо, который часто называют проходом.

**3 Алгоритм Дейкстры**

Исследованию формальных способов преобразования арифметических и логических выражений в ОПЗ посвящены многочисленные исследования, однако в практике системного программирования наибольшее распространение получили способы преобразования на основе алгоритма Дейкстры.

Суть алгоритма Дейкстры можно представить следующим рисунком:

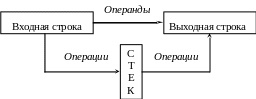


Рисунок 1 – Суть алгоритма Дейкстры

Из этого рисунка следует, что на вход алгоритма посимвольно поступает исходное выражение. Операнды исходного выражения пропускаются на выход и формируют так же посимвольно выходную строку. Операции обрабатываются по определенным правилам на основе стека.

Для реализации такой обработки известное в системном программировании понятие стека используется также в алгоритме Дейкстры для размещения в нем операций. При этом предварительно каждой операции приписывается свой приоритет на основе таблицы приоритетов, которая приведена ниже (таблица 2).

Таблица 2 – Таблица приоритетов

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| (  for  if  while  [  АЭМ  Ф | 0 |
| )  ,  do  else  elif  ] | 1 |
| = | 2 |
| or | 3 |
| and | 4 |
| not | 5 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| <  <=  !=  =  >  >= | 6 |
| +  -  +=  -=  \*=  /=  \*\*=  %= | 7 |
| \*  /  % | 8 |
| return | 9 |

**4 Перевод операторов цикла в ОПЗ**

Обработка оператора цикла с предусловием WHILE выражение DO оператор;:

1. Символ WHILE из входной строки заносится в стек. В стеке к символу WHILE добавляется рабочая метка Mi и после этого в выходную строку записывается часть Mi:.
2. Символ DO выталкивает в выходную строку все операции из стека до ближайшего WHILE Mi. В стеке к WHILE Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается часть Mi+1 УПЛ.
3. Символ ‘;’ указывает на конец оператора цикла с предусловием и выталкивает из стека все символы до ближайшего WHILE Mi Mi+1, при этом сам WHILE уничтожается, а в выходную строку помещается Mi БП Mi+1:.

Обработка оператора цикла с постусловием REPEAT оператор UNTIL выражение; можно заменить последовательностью операторов

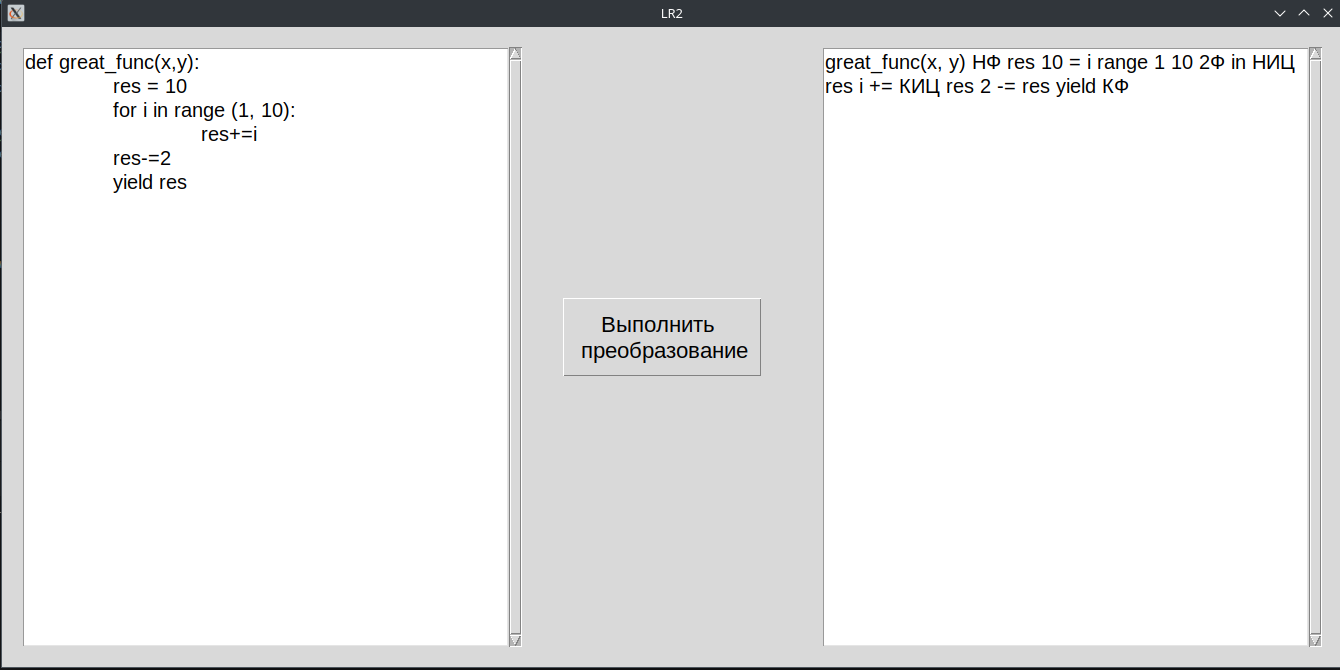
1. Символ ‘;’ указывает на конец оператора цикла с постусловием и выталкивают из стека все символы до ближайшего REPEAT Mi, при этом сам REPEAT уничтожается, а в выходную строку помещаются Mi+1 УПЛ Mi БП Mi+1:.

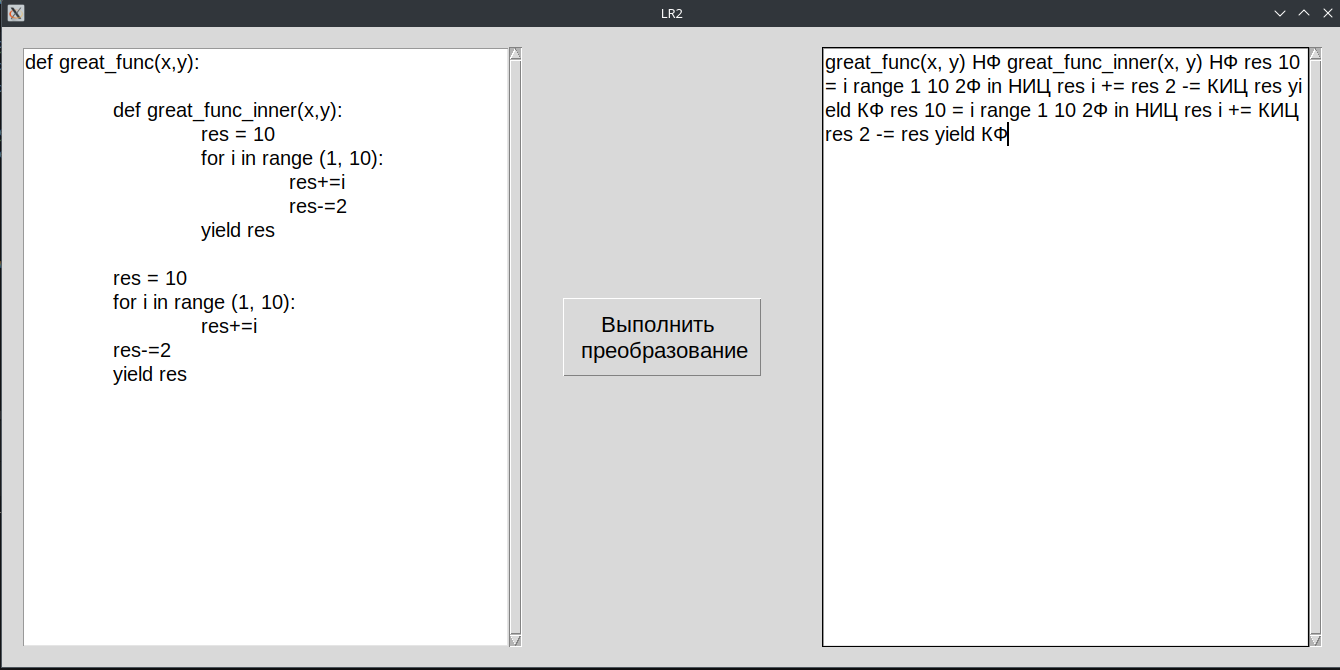
Обработка оператора цикла с счетчиком FOR переменная = значение1 TO значение2 DO оператор;:

1. FOR переменная из входной строки заносится в стек.
2. Символ TO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная. В стеке к FOR переменная добавляется рабочая метка Mi и после этого в выходную строку записывается Mi: переменная.
3. Символ DO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная Mi. В стеке к FOR переменная Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается <= Mi+1 УПЛ.
4. Символы ‘;’ и END указывают на конец оператора цикла с счетчиком и выталкивают из стека все символы до ближайшего FOR переменная Mi Mi+1, при этом сам FOR уничтожается, а в выходную строку помещается метка переменная переменная 1 + := Mi БП Mi+1:.

**5 Результаты экспериментов**

Примеры работы программы для программы для тестирования.

Рисунок 2 – Результат работы программы

Рисунок 3 – Результат работы программы

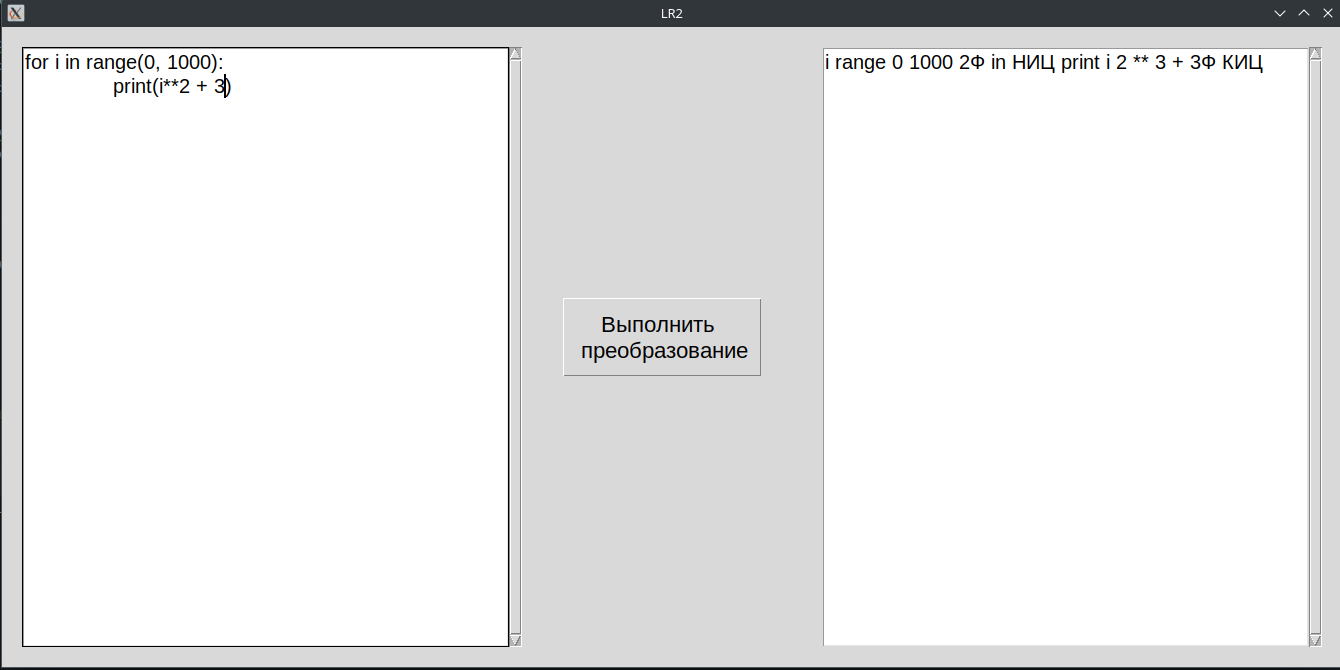


Рисунок 4 – Результат работы программы

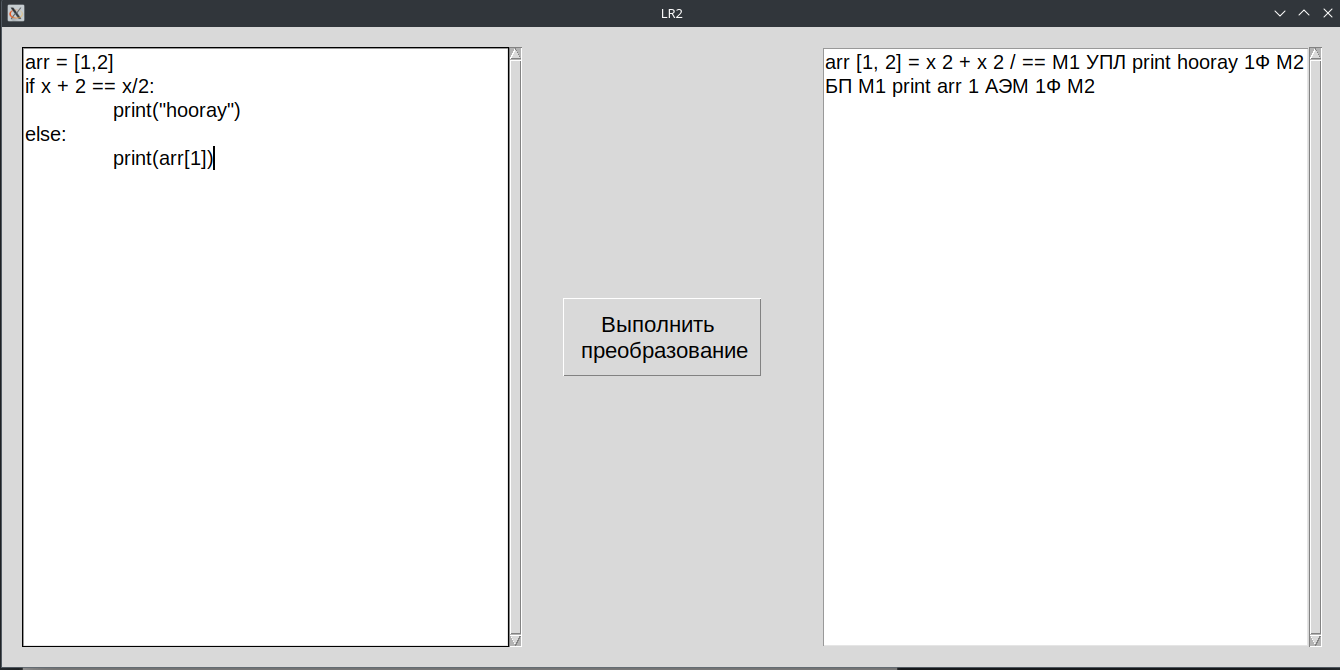


Рисунок 5 – Результат работы программы

**ПРИЛОЖЕНИЕ А   
Листинг программы**

import ast

from \_ast import Module, AST

from tkinter import \*

import tkinter.scrolledtext as st

binaryOperationNameToSymbol = {

'Mult': '\*',

'Add': '+',

'Sub': '-',

'Div': '/',

'FloorDiv': '//',

'Mod': '%',

'Pow': '\*\*',

}

compareEqNameToSymbol = {

'Lt': '<',

'LtE': '<=',

'Gt': '>',

'GtE': '>=',

'Eq': '==',

'NotEq': '!=',

}

compareBoolNameToSymbol = {

'And': 'and',

'Or': 'or',

}

compareUnaryNameToSymbol = {

'Not': 'not',

}

compareAugAssignNameToSymbol = {

'Add': '+=',

'Sub': '-=',

'Mult': '\*=',

'Div': '/=',

'MatMult': '\*\*='

}

def calcTreeNodes(node: AST | Module, result=0) -> int:

if isinstance(node, list):

return result + sum([calcTreeNodes(node, 0) for node in node])

elif hasattr(node, 'left'):

return result + calcTreeNodes(node.left, 0) + calcTreeNodes(node.right, 0)

elif hasattr(node, 'values'):

return result + sum([calcTreeNodes(node, 0) for node in node.values])

else:

return result + 1

def convert\_to\_rpn(node):

if isinstance(node, ast.BinOp):

left = convert\_to\_rpn(node.left)

right = convert\_to\_rpn(node.right)

return left + right + binaryOperationNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.Num):

return str(node.n) + " "

elif isinstance(node, ast.Expr):

return convert\_to\_rpn(node.value)

elif isinstance(node, ast.FunctionDef):

args = ", ".join([arg.arg for arg in node.args.args])

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return node.name + "(" + args + ") " + "НФ " + body + "КФ "

elif isinstance(node, ast.Assign):

target = convert\_to\_rpn(node.targets[0])

value = convert\_to\_rpn(node.value)

return target + value + "= "

elif isinstance(node, ast.Compare):

left = convert\_to\_rpn(node.left)

ops = " ".join([compareEqNameToSymbol.get(op.\_\_class\_\_.\_\_name\_\_) for op in node.ops])

comparators = " ".join([convert\_to\_rpn(comp) for comp in node.comparators])

return left + comparators + ops + " "

elif isinstance(node, ast.Return):

value = convert\_to\_rpn(node.value)

return value + "return "

elif isinstance(node, ast.Yield):

value = convert\_to\_rpn(node.value)

return value + "yield "

elif isinstance(node, ast.AugAssign):

target = convert\_to\_rpn(node.target)

value = convert\_to\_rpn(node.value)

return target + value + compareAugAssignNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.For):

target = convert\_to\_rpn(node.target)

print(node.target)

iter = convert\_to\_rpn(node.iter)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return target + iter + "in " + "НИЦ " + body + "КИЦ "

elif isinstance(node, ast.While):

test = convert\_to\_rpn(node.test)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

return test + "НУЦ " + body + "КУЦ "

elif isinstance(node, ast.List):

elts = ", ".join([convert\_to\_rpn(elt).strip() for elt in node.elts])

return "[" + elts + "] "

elif isinstance(node, ast.Dict):

keys = [convert\_to\_rpn(key) for key in node.keys]

values = [convert\_to\_rpn(value) for value in node.values]

key\_value\_pairs = [k + ": " + v for k, v in zip(keys, values)]

pairs\_str = ", ".join(key\_value\_pairs)

return "{ " + pairs\_str + "} "

elif isinstance(node, ast.Attribute):

value = convert\_to\_rpn(node.value)

attr = node.attr

return value + attr + " "

elif isinstance(node, ast.Num):

return str(node.n) + " "

elif isinstance(node, ast.Call):

args = [convert\_to\_rpn(arg) for arg in node.args]

return node.func.id + " " + "".join(args) + str(calcTreeNodes(node.args)) + "Ф "

elif isinstance(node, ast.Expr):

return convert\_to\_rpn(node.value)

elif isinstance(node, ast.If):

test = convert\_to\_rpn(node.test)

body = " ".join([convert\_to\_rpn(n) for n in node.body])

if len(node.orelse) == 0:

return test + "M1 УПЛ " + body + "М1 "

orelse = " ".join([convert\_to\_rpn(n) for n in node.orelse])

return test + "M1 УПЛ " + body + "М2 БП М1 " + orelse + "М2 "

elif isinstance(node, ast.Subscript):

value = convert\_to\_rpn(node.value)

slice\_value = convert\_to\_rpn(node.slice)

return value + slice\_value + "АЭМ "

elif isinstance(node, ast.Slice):

lower = convert\_to\_rpn(node.lower) if node.lower is not None else ""

upper = convert\_to\_rpn(node.upper) if node.upper is not None else ""

step = convert\_to\_rpn(node.step) if node.step is not None else ""

return lower + upper + step + "SLICE "

elif isinstance(node, ast.UnaryOp):

operand = convert\_to\_rpn(node.operand)

return operand + compareUnaryNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

elif isinstance(node, ast.Name):

return node.id + " "

elif isinstance(node, ast.BoolOp):

if len(node.values) == 2:

left = convert\_to\_rpn(node.values[0])

right = convert\_to\_rpn(node.values[1])

return left + right + compareBoolNameToSymbol.get(node.op.\_\_class\_\_.\_\_name\_\_) + " "

else:

print(node.op, node.values)

elif isinstance(node, ast.Constant):

return node.value + " "

elif isinstance(node, ast.IfExp):

test = convert\_to\_rpn(node.test)

body = convert\_to\_rpn(node.body)

orelse = convert\_to\_rpn(node.orelse)

return test + "M1 УПЛ " + body + "М2 БП М1 " + orelse + "М2 "

else:

print(node)

return ""

def python\_to\_rpn(source\_code):

tree = ast.parse(source\_code)

rpn\_expression = ""

for node in tree.body:

rpn\_expression += convert\_to\_rpn(node)

return rpn\_expression.strip().replace(' ', ' ')

def prog():

f = open('./resources/python.txt', 'r')

input\_sequence = f.read()

f.close()

out\_seq = python\_to\_rpn(input\_sequence)

# файл, содержащий обратную польскую запись

f = open('gen/rpn.txt', 'w')

f.write(out\_seq)

f.close()

def write\_txt(data, to):

with open(to, 'w') as file:

file.write(data)

def clicked():

write\_txt(codetxt.get("1.0", "end"), 'resources/python.txt')

opzstext.delete("1.0", END)

prog()

f1 = open('gen/rpn.txt', 'r')

text = f1.read()

opzstext.insert("1.0", text)

f1.close()

window = Tk()

window.title("LR2")

f1 = open('resources/python.txt', 'r')

text = f1.read()

window.geometry('1340x640')

codetxt = st.ScrolledText(window, font=("Arial", 18))

codetxt.insert("1.0", text)

codetxt.place(x=20, y=20, width=500, height=600)

opzstext = st.ScrolledText(window, font=("Arial", 18))

opzstext.place(x=820, y=20, width=500, height=600)

btngo = Button(window, text="Выполнить \n преобразование", command=clicked, font=("Arial", 20))

btngo.place(x=510 + 150 - 100, y=10 + 300 - 40, width=200, height=80)

window.mainloop()