**Федеральное государственное образовательное**

**бюджетное учреждение высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**ИНСТИТУТ ЗАОЧНОГО И ОТКРЫТОГО ОБРАЗОВАНИЯ**

**ДЕПАРТАМЕНТ АНАЛИЗА ДАННЫХ И МАШИННОГО ОБУЧЕНИЯ**

**ПРАКТИЧЕСКАЯ РАБОТА № 2**

**по дисциплине «Практикум по программированию»**

**Текстовый калькулятор**

Направление подготовки - 09.03.03 «Прикладная информатика»

Студент(ка) **Белоусов А.А.**

(Ф.И.О.)

Курс2 № группы **ЗБ-ПИ19-1**

Преподаватель **Горохова Р.И.**

(Ф.И.О.)

Москва 2020

Задание

Базовая часть (выполняется всеми самостоятельно!):

Написать калькулятор для строковых выражений вида '<число> <операция> <число>', где <число> - не отрицательное целое число меньшее 100, записанное словами, например "тридцать четыре", <арифметическая операция> - одна из операций "плюс", "минус", "умножить". Результат выполнения операции вернуть в виде текстового представления числа. Пример calc("двадцать пять плюс тринадцать") -> "тридцать восемь"

Оформить калькулятор в виде функции, которая принимает на вход строку и возвращает строку.

Дополнительные задания

1) Реализовать поддержку операции деления

3) Реализовать текстовый калькулятор для выражения из произвольного количества операций с учетом приоритета операций. Пример: calc("пять плюс два умножить на три минус один") -> "десять". (Для реализации рекомендуется использовать алгоритмы основанные на польской инверсной записи)

**Сложность 3**

4) Расширение задания 3. Добавить поддержку приоритета операций с помощью скобок. Пример: calc("скобка открывается пять плюс два скобка закрывается умножить на три минус один") -> "двадцать".

**Сложность 3**

5) Добавить возможность использования отрицательных чисел. Пример: calc("пять минус минус один") -> "шесть".

**Сложность 1**

6) Добавить возможность оперировать с дробями (правильными и смешанными). Реализовать поддержку сложения, вычитания и умножения, дробей. Результат операций не должен представлять неправильную дробь, такие результаты нужно превращать в смешанные дроби. Пример: calc("один и четыре пятых плюс шесть седьмых ") -> "два и двадцать три тридцать пятых".

**Сложность 3**

7) Расширение задания 6. Добавить автоматическое сокращение дроби в ответе. Пример: calc("одна шестая умножить на две третьих") -> "одна девятая".

**Сложность 1**

10) Диагностировать ошибки: неправильную запись числа; неправильную последовательность чисел и операций; (задание 1) деление на ноль; (задание 3) неправильную последовательность чисел и операций; (задание 4) некорректный баланс и вложенность скобок; (задание 6) некорректную запись числа

**Сложность 1 или 2**

Код программы

Код файла main.py

import re  
from Fraction import Fraction  
from calculator import evaluatenumbers = {**"ноль"**: **'0'**, **"один"**: **'1'**, **"одна"**: **'1'**, **"два"**: **'2'**, **"две"**: **'2'**, **"три"**: **'3'**, **"четыре"**: **'4'**, **"пять"**: **'5'**,  
 **"шесть"**: **'6'**, **"семь"**: **'7'**,  
 **"восемь"**: **'8'**, **"девять"**: **'9'**, **"десять"**: **'10'**, **"одиннадцать"**: **'11'**, **"двенадцать"**: **'12'**,  
 **"тринадцать"**: **'13'**, **"четырнадцать"**: **'14'**, **"пятнадцать"**: **'15'**, **"шестнадцать"**: **'16'**, **"семнадцать"**: **'17'**,  
 **"восемнадцать"**: **'18'**, **"девятнадцать"**: **'19'**, **"двадцать"**: **'20'**, **"тридцать"**: **'30'**, **"сорок"**: **'40'**,  
 **"пятьдесят"**: **'50'**, **"шестьдесят"**: **'60'**, **"семьдесят"**: **'70'**, **"восемьдесят"**: **'80'**, **"девяносто"**: **'90'**}  
operators = {**"плюс"**: **'+'**, **"минус"**: **'-'**, **"умножить на"**: **'\*'**, **"разделить на"**: **"/"**, **"и"**: **"?"**,  
 **"скобка открывается"**: **"("**, **"скобка закрывается"**: **")"**}  
denominators = {**"первых"**: **'1'** + **' #'**, **"вторых"**: **'2'** + **' #'**, **"третьих"**: **'3'** + **' #'**, **"четвёртых"**: **'4'** + **' #'**,  
 **"пятых"**: **'5'** + **' #'**, **"шестых"**: **'6'** + **' #'**,  
 **"седьмых"**: **'7'** + **' #'**, **"восьмых"**: **'8'** + **' #'**, **"девятых"**: **'9'** + **' #'**, **"десятых"**: **'10'** + **' #'**,  
 **"одиннадцатых"**: **'11'** + **' #'**,  
 **"двенадцатых"**: **'12'** + **' #'**, **"тринадцатых"**: **'13'** + **' #'**, **"четырнадцатых"**: **'14'** + **' #'**,  
 **"пятнадцатых"**: **'15'** + **' #'**,  
 **"шестнадцатых"**: **'16'** + **' #'**, **"семнадцатых"**: **'17'** + **' #'**, **"восемнадцатых"**: **'18'** + **' #'**,  
 **"девятнадцатых"**: **'19'** + **' #'**,  
 **"двадцатых"**: **'20'** + **' #'**, **"тридцатых"**: **'30'** + **' #'**, **"сороковых"**: **'40'** + **' #'**,  
 **"пятидесятых"**: **'50'** + **' #'**, **"шестидесятых"**: **'60'** + **' #'**,  
 **"семидесятых"**: **'70'** + **' #'**, **"восьмидесятых"**: **'80'** + **' #'**, **"девяностая"**: **'90'** + **' #'**,  
 **"первая"**: **'1'** + **' #'**, **"вторая"**: **'2'** + **' #'**, **"третья"**: **'3'** + **' #'**, **"четвёртая"**: **'4'** + **' #'**,  
 **"пятая"**: **'5'** + **' #'**, **"шестая"**: **'6'** + **' #'**,  
 **"седьмая"**: **'7'** + **' #'**, **"восьмая"**: **'8'** + **' #'**, **"девятая"**: **'9'** + **' #'**, **"десятая"**: **'10'** + **' #'**,  
 **"одиннадцатая"**: **'11'** + **' #'**,  
 **"двенадцатая"**: **'12'** + **' #'**, **"тринадцатая"**: **'13'** + **' #'**, **"четырнадцатая"**: **'14'** + **' #'**,  
 **"пятнадцатая"**: **'15'** + **' #'**,  
 **"шестнадцатая"**: **'16'** + **' #'**, **"семнадцатая"**: **'17'** + **' #'**, **"восемнадцатая"**: **'18'** + **' #'**,  
 **"девятнадцатая"**: **'19'** + **' #'**,  
 **"двадцатая"**: **'20'** + **' #'**, **"тридцатая"**: **'30'** + **' #'**, **"сороковая"**: **'40'** + **' #'**,  
 **"пятидесятая"**: **'50'** + **' #'**, **"шестидесятая"**: **'60'** + **' #'**,  
 **"семидесятая"**: **'70'** + **' #'**, **"восьмидесятая"**: **'80'** + **' #'**, **"девяностых"**: **'90'** + **' #'** }  
  
  
def getKeyByValue(dict, value):  
 return list(dict.keys())[list(dict.values()).index(value)]  
  
  
def wordsToExpression(text):  
 text = text.lower()if bool(re.search(**r"[^а-яё ]"**, text)):  
 raise SyntaxError(**"В исходном выражении содержаться недопустимые символы"**)words = dict(numbers)  
 words.update(operators)  
 words.update(denominators)  
  
 for word, value in words.items():text = re.sub(**fr"\b**{word}**\b"**, **' '** + value, text)  
if bool(re.search(**r"[^+\-\*/#()?0-9 ]"**, text)):  
 raise SyntaxError(**"В исходном выражении содержаться недопустимые слова"**)  
  
 if not balanced(text):  
 raise SyntaxError(**"Скобки раставлены неверно"**)  
  
 return list(filter(None, text.split(**' '**)))  
  
  
def isInt(value):  
 try:  
 return int(value), True  
 except ValueError:  
 return value, False  
  
  
def accumulateNumbers(expressionList):  
 i = 0  
 sumFlag = False  
 while (i < len(expressionList)):  
 (value, isNumber) = isInt(expressionList[i])  
 if isNumber:  
 expressionList[i] = value  
 if value >= 20 and value % 10 == 0:  
 sumFlag = True  
 if sumFlag and value < 10:expressionList[i - 1:i + 1] = [expressionList[i - 1] + expressionList[i]]  
 i -= 1sumFlag = False  
 else:  
 sumFlag = False  
 i += 1  
  
 return expressionList  
  
  
def numberExpressionToFractions(numberExpressionList):  
 numberExpressionList.append(**"f"**)i = 0mixedFlag = False  
 fraction = []  
  
 def createFraction():  
 nonlocal fraction  
 nonlocal mixedFlag  
 nonlocal i  
 if fraction:  
 if mixedFlag and len(fraction) == 3:  
 frac = Fraction(fraction[0] \* fraction[-1] + fraction[1], fraction[-1])  
 numberExpressionList[i - 4:i + 1] = [frac]  
 i -= 4  
 mixedFlag = False  
 elif not mixedFlag:  
 if numberExpressionList[i] == **"#"**:  
 if len(fraction) == 2:  
 frac = Fraction(fraction[0], fraction[1])  
 numberExpressionList[i - 2:i + 1] = [frac]  
 i -= 2  
 else:  
 raise SyntaxError(**"Ошибка в записи числа"**)  
 else:  
 if len(fraction) == 1:  
 frac = Fraction(fraction[0], 1)  
 numberExpressionList[i - 1] = frac  
 else:  
 raise SyntaxError(**"Ошибка в записи числа"**)  
 else:  
 raise SyntaxError(**"Ошибка в записи числа"**)  
 fraction = []  
  
 while (i < len(numberExpressionList)):  
 if isinstance(numberExpressionList[i], int):fraction.append(numberExpressionList[i])elif numberExpressionList[i] == **"?"**:  
 if mixedFlag:  
 raise SyntaxError(**"Дублирование союза 'и' в выражении"**)  
 mixedFlag = True  
 else:createFraction()  
 i += 1  
numberExpressionList.pop()  
  
 return numberExpressionList  
  
  
def checkFractionExpression(fractionExpression):lastEl = **"s"** canDoublePlusFlag = False  
 canDoubleMinusFlag = False  
 for i in range(len(fractionExpression)):if isinstance(fractionExpression[i], Fraction):  
 if lastEl in **"n)"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 lastEl = **"n"** elif fractionExpression[i] in **"\*/"**:  
 *# n)* if lastEl in **"s(\*/+-"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 lastEl = fractionExpression[i]  
 elif fractionExpression[i] == **"+"**:  
 if lastEl in **"\*/-"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 if lastEl in **"n)"**:  
 canDoublePlusFlag = True  
 elif lastEl == **"+"**:  
 if canDoublePlusFlag:  
 canDoublePlusFlag = False  
 else:  
 raise SyntaxError(**"Недопустимый повтор +"**)  
 lastEl = fractionExpression[i]  
 elif fractionExpression[i] == **"-"**:  
 if lastEl in **"\*/+"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 if lastEl in **"n)"**:  
 canDoubleMinusFlag = True  
 elif lastEl == **"-"**:  
 if canDoubleMinusFlag:  
 canDoubleMinusFlag = False  
 else:  
 raise SyntaxError(**"Недопустимый повтор -"**)  
 lastEl = fractionExpression[i]  
 elif fractionExpression[i] == **"("**:  
 if lastEl in **"n)"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 lastEl = **"("** elif fractionExpression[i] == **")"**:  
 if lastEl in **"s\*/+-"**:  
 raise SyntaxError(**"Ошибка порядка элементов в выражении"**)  
 lastEl = **")"** else:  
 raise SyntaxError(**"Ошибка дублирования элементов в выражении"**)  
  
 if lastEl in **"\*/-+(s"**:  
 raise SyntaxError(**"Выражение не завершенно"**)  
  
  
def balanced(s):  
 pairs = {**"{"**: **"}"**, **"("**: **")"**, **"["**: **"]"**}  
 stack = []  
 for c in s:  
 if c in **"{[("**:  
 stack.append(c)  
 elif c in **"}])"**:  
 if stack and c == pairs[stack[-1]]:  
 stack.pop()  
 else:  
 return False  
 return len(stack) == 0  
  
  
def uintToWord(number, words, den=False):  
 if number > 20 and number % 10:  
 return getKeyByValue(words, str(number - number % 10)) + **" "** + getKeyByValue(words, str(  
 number % 10) + (**" #"** if den else **""**))  
 else:  
 return getKeyByValue(words, str(number) + (**" #"** if den else **""**))  
  
  
def fractionToStr(frac):  
 result = **""**(val, num, den) = frac.getElements()  
 if any(map(lambda el: abs(el) >= 100, frac.getElements())):  
 if den == 1:  
 return str(val)  
 if val:  
 result = str(val) + **" и "** result += str(num) + **" / "** + str(den)  
 else:  
 if den == 1:  
 if val < 0:  
 result = **"минус "** val \*= -1  
 result += uintToWord(val, numbers)  
 return result  
 if val:  
 if val < 0:  
 result = **"минус "** val \*= -1  
  
 result += uintToWord(val, numbers) + **" и "** if num < 0:  
 result = **"минус "** num \*= -1  
 result += uintToWord(num, numbers) + **" "** words = dict(numbers)  
 words.update(denominators)  
 result += uintToWord(den, words, True)  
  
 return result  
  
  
def evalExpression(expression):  
 lstExp = numberExpressionToFractions(accumulateNumbers(wordsToExpression(expression)))  
 checkFractionExpression(lstExp)  
 result = evaluate(lstExp)  
 return fractionToStr(result)  
  
  
def calculator(expression):  
 print(expression)  
 try:  
 print(**" = "** + evalExpression(expression))  
 except Exception as e:  
 print(**"Ошибка: "** + str(e))  
  
  
def main():print(**"Допустимые выражения: "**)  
 calculator(evalExpression(  
 **"скобка открывается плюс пять и одна вторая плюс скобка открывается минус семь скобка закрывается умножить на пять скобка закрывается разделить на одна вторая плюс пять третьих"**))  
 calculator(  
 **"скобка открывается плюс пять и одна вторая плюс плюс семь и семь седьмых плюс скобка открывается минус семь скобка закрывается умножить на пять скобка закрывается разделить на одна вторая плюс пять третьих"**)  
 calculator(**" пять плюс семь "**)  
 calculator(**"скобка открывается скобка открывается пять плюс семь скобка закрывается скобка закрывается"**)  
 calculator(**"пять умножить на скобка открывается плюс семь скобка закрывается"**)  
 calculator(**"пять минус минус семь"**)  
 calculator(**"пять плюс плюс семь"**)  
 calculator(**"пять плюс скобка открывается минус семь скобка закрывается"**)  
 calculator(**"минус семь плюс семь и одна четвёртая"**)  
 calculator(**"девяносто плюс девяносто"**)  
 calculator(**"девяносто и одна четвёртая плюс девяносто и одна пятая"**)  
 calculator(**"минус пять девяностых плюс семь пятидесятых"**)  
 longStr = [**"скобка открывается скобка открывается"** +  
 **" скобка открывается минус пять плюс пять и одна четвёртая скобка закрывается"**+  
 **" умножить на скобка открывается минус пять и одна восьмая плюс плюс семь и "**+  
 **"две четвёртых скобка закрывается плюс скобка открывается минус ноль умножить на "**+  
 **"скобка открывается плюс семь и одна пятидесятая скобка закрывается разделить на "**+  
 **"скобка открывается минус семь девятых скобка закрывается скобка закрывается скобка закрывается"** +**" плюс скобка открывается плюс семь четвёртых умножить на скобка открывается плюс пять вторых "**+  
 **"скобка закрывается скобка закрывается скобка закрывается разделить на скобка открывается "**+  
 **"пятнадцать и шестнадцать четвёртых умножить на семь вторых скобка закрывается умножить на "**+  
 **"семь и одна четвёртая разделить на скобка открывается минус четыре скобка закрывается "** +**"плюс семь умножить на два и пять вторых"**]calculator(\*longStr)  
print(**"Некорректные выражения"**)  
 calculator(**"- пять плюс семь"**)  
 calculator(**"скобка открывается пять плюс семь"**)  
 calculator(**"скобка открывается скобка открывается пять плюс семь скобка закрывается"**)  
 calculator(**"скобка закрывается пять плюс семь скобка открывается"**)  
 calculator(**"пять плюс семь плюс"**)  
 calculator(**"плюс плюс пять плюс семь"**)  
 calculator(**"минус минус пять плюс семь"**)  
 calculator(**"пять умножить семь"**)  
 calculator(**"пять умножить на плюс семь"**)  
 calculator(**"пять плюс минус семь"**)  
 calculator(**"пять скобка открывается плюс скобка закрывается минус семь"**)  
 print()  
 while True:  
 try:  
 print(**"Введите выражение: "**)  
 calculator(input())  
 except KeyboardInterrupt:  
 exit(0)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

Код файла calculator.py

from Fraction import Fraction  
  
  
def is\_op(el):  
 return el in **"+-\*/"**def priority(op):  
 if **"u"** in op:  
 return 3  
 if op in **"+-"**:  
 return 1  
 elif op in **"\*/"**:  
 return 2  
 return -1  
  
  
def process\_op(stack, op):  
 if **"u"** in op:  
 l = stack.pop()  
 {  
 **"u+"**: lambda: stack.append(l),  
 **"u-"**: lambda: stack.append(Fraction(-l.getNumerator(),l.getDenominator())),  
 }[op]()  
 else:  
 r = stack.pop()  
 l = stack.pop()  
 {  
 **'+'**: lambda: stack.append(l + r),  
 **'-'**: lambda: stack.append(l - r),  
 **'\*'**: lambda: stack.append(l \* r),  
 **'/'**: lambda: stack.append(l / r)  
 }[op]()  
  
  
def evaluate(expressionList):  
 numbers = []  
 operators = []  
 may\_be\_unary = True  
 for i in range(len(expressionList)):  
 if isinstance(expressionList[i], Fraction):  
 numbers.append(expressionList[i])  
 may\_be\_unary = False  
 elif expressionList[i] == **'('**:  
 operators.append(**'('**)  
 may\_be\_unary = True  
 elif expressionList[i] == **')'**:  
 while (operators[-1] != **'('**):  
 process\_op(numbers, operators.pop())  
operators.pop()  
 may\_be\_unary = False  
 else:  
 if may\_be\_unary and expressionList[i] in **"+-"**:  
 expressionList[i] = **"u"** + expressionList[i]  
while (operators and (  
 (len(expressionList[i]) == 1 and priority(operators[-1]) >= priority(expressionList[i])) or  
 (len(expressionList[i]) == 2 and priority(operators[-1]) > priority(expressionList[i])))):  
 process\_op(numbers, operators.pop())  
 operators.append(expressionList[i])  
 may\_be\_unary = True  
while (operators):  
 process\_op(numbers, operators.pop())  
  
 return numbers[-1]

Код файла Fraction.py

class Fraction:  
 def \_reduce(self, numerator, denominator):  
 if (numerator < 0 and denominator >= 0) or (numerator >= 0 and denominator < 0):  
 sign = -1  
 else:  
 sign = 1  
  
 a = abs(numerator)  
 b = abs(denominator)  
  
 i = min(a, b)  
 if (i == 0):  
 return  
  
 while ((a % i != 0) or (b % i != 0)):  
 i -= 1  
  
 a //= i  
 b //= i  
  
 return (a \* sign, b)  
  
 def getNumerator(self):  
 return self.\_numerator  
  
 def getDenominator(self):  
 return self.\_denominator  
  
 def getElements(self):  
 if (self.\_numerator < 0 and self.\_denominator >= 0) or (self.\_numerator >= 0 and self.\_denominator < 0):  
 sign = -1  
 else:  
 sign = 1  
  
 num = abs(self.\_numerator)  
 den = abs(self.\_denominator)  
 val = num // den  
 if val:  
 num%=den  
 val\*=sign  
 else:  
 num\*=sign  
  
 return (val, num, den)  
  
  
 def \_\_init\_\_(self, numerator, denominator):  
 if (not isinstance(numerator, int)):  
 raise TypeError(**"The numerator of a Faction must be an integer"**)  
 if (not isinstance(denominator, int)):  
 raise TypeError(**"The denominator of a Faction must be an integer"**)  
  
 if (denominator == 0):  
 raise ZeroDivisionError(**"Знаменатель дроби не может быть нулём"**)  
  
 if (numerator == 0):  
 self.\_numerator = 0  
 self.\_denominator = 1  
 else:(self.\_numerator, self.\_denominator) = self.\_reduce(numerator, denominator)  
  
 def \_\_str\_\_(self):  
 return str(self.\_numerator) + **"/"** + str(self.\_denominator)  
  
 def \_\_eq\_\_(self, other):  
 return (self.getNumerator() == other.getNumerator()) and (self.getDenominator() == other.getDenominator())  
  
 def \_\_ne\_\_(self, other):  
 return not self == other  
  
 def \_\_lt\_\_(self, other):  
 return self.getNumerator() \* other.getDenominator() < self.getDenominator() \* other.getNumerator()  
  
 def \_\_le\_\_(self, other):  
 return not other < self  
  
 def \_\_gt\_\_(self, other):  
 return other < self  
  
 def \_\_ge\_\_(self, other):  
 return not other > self  
  
 def \_\_add\_\_(self, other):  
 num = self.getNumerator() \* other.getDenominator() + other.getNumerator() \* self.getDenominator()  
 den = self.getDenominator() \* other.getDenominator()  
 return Fraction(num, den)  
  
 def \_\_sub\_\_(self, other):  
 num = self.getNumerator() \* other.getDenominator() - other.getNumerator() \* self.getDenominator()  
 den = self.getDenominator() \* other.getDenominator()  
 return Fraction(num, den)  
  
 def \_\_mul\_\_(self, other):  
 num = self.getNumerator() \* other.getNumerator()  
 den = self.getDenominator() \* other.getDenominator()  
 return Fraction(num, den)  
  
 def \_\_truediv\_\_(self, other):  
 num = self.getNumerator() \* other.getDenominator()  
 den = self.getDenominator() \* other.getNumerator()  
 return Fraction(num, den)

Результат выполнения программы

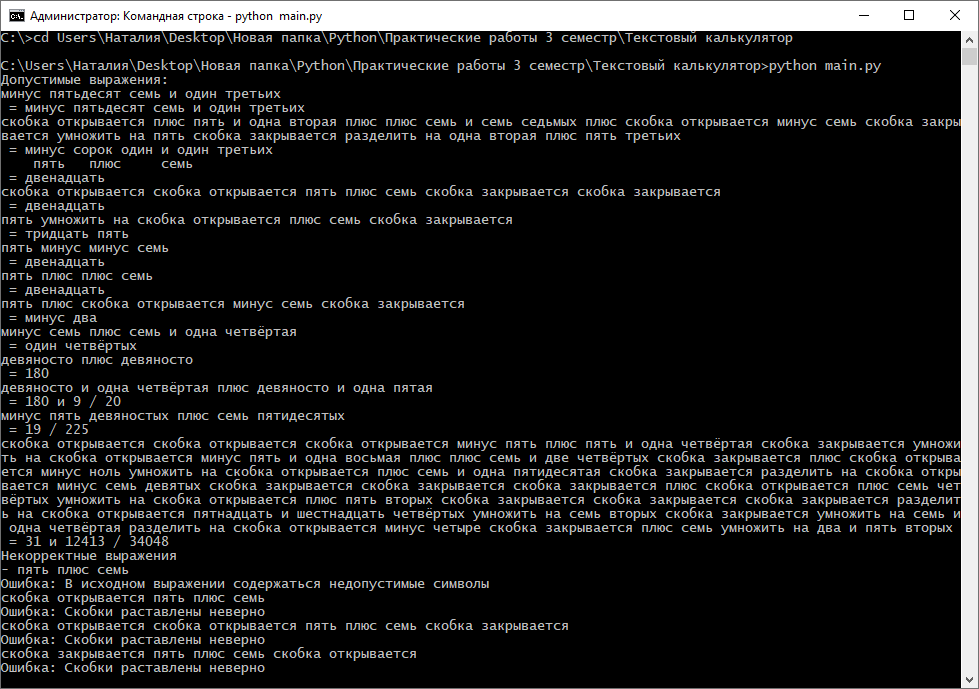


Рис.1 – Стартовый экран программы

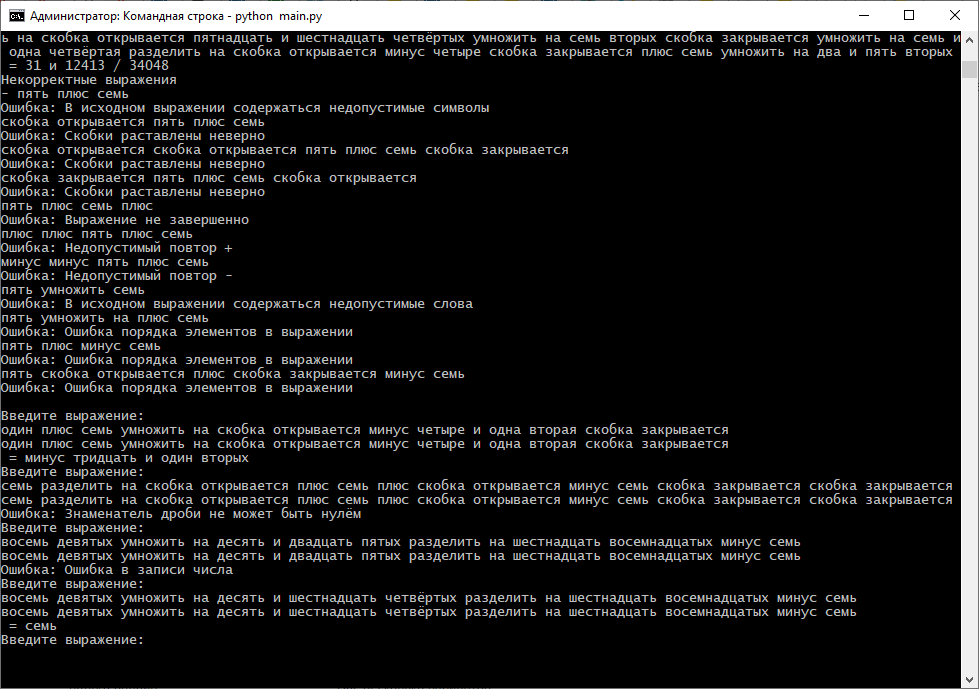


Рис.2 – Результат вычисления выражений программой