**Отчет должен содержать**

1. Титульный лист с указанием варианта.
2. Постановка задачи.
3. Обоснование выбора используемых структур данных.
4. Описание алгоритма решения.
5. Пример работы.
6. Листинг.

Покрытие кода unit-тестами и обработка исключительных ситуаций являются ***обязательными*** требованиями.

Реализуемая программа должна корректно реагировать на неправильные входные данные, выводить сообщения об ошибках.

При выполнении курсовой работы можно использовать **только классы и структуры, написанные самостоятельно**.

**Преобразование алгебраических формул из инфиксной в префиксную форму записи и вычисление значения выражения**

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. Программа должна выполнить проверку корректности введенного выражения. В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода. В противном выводится польская нотация введенного выражения, а также отображается результат вычисления.

*Входные данные:*

* арифметическое выражение
* поддерживаемый тип данных: вещественные числа (double)
* поддерживаемые знаки: +, -, \*, /, ^, унарный “-”, функции с одним аргументом (cos, sin, tg, ctg, ln, log, sqrt др. (хотя бы одну не из списка)abs), константы pi, e открывающая и закрывающая скобки

*Выходные данные:*

* префиксная ФЗ
* результат вычисления

Входные данные по желанию можно читать из файла.

Инфиксная нотация: [https://ru.wikipedia.org/wiki/Инфиксная\_нотация](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D0%B0%D1%8F_%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F)

Префиксная нотация: [https://ru.wikipedia.org/wiki/Польская\_нотация](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F)

**Постановка задачи.**

Реализовать калькулятор.

Ввод математического выражения, состоящего из чисел и арифметических знаков из консоли.

Проверка корректности введенного выражения.

В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода.

В противном вывести польскую нотацию введенного выражения, а также результат вычисления.

*Входные данные:*

* арифметическое выражение
* поддерживаемый тип данных: вещественные числа (double)
* поддерживаемые знаки: +, -, \*, /, ^, унарный “-”, функции с одним аргументом (cos, sin, tg, ctg, ln, log, sqrt), константы pi, e открывающая и закрывающая скобки
* В качестве дополнительной функции с одним аргументом выбрана abs

*Выходные данные:*

* префиксная ФЗ
* результат вычисления

**Обоснование выбора используемых структур данных.**

Для реализации алгоритма преобразования инфиксной формы записи в префиксную используется структура данных стек:

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def is\_empty(self):  
 return self.items == []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()  
  
 def peek(self):  
 return self.items[len(self.items) - 1]  
  
 def size(self):  
 return len(self.items)

String: Для хранения вводимого пользователем математического выражения состоящего из чисел и арифметических знаков

Dictionary: Для сопоставления операций и их приоритетов

List: Для хранения преобразованного математического выражения

**Описание алгоритма решения.**

Преобразование инфиксной формы записи в префиксную:

1. Инфиксная форма записи преобразуется в list, содержащий введенные числа, операции и скобки
2. Список с инфиксным выражением записывается справа налево и последовательно обрабатывается:
   1. Если текущий элемент число, то добавляется в очередь.
   2. Если входящий элемент является левой скобкой, то он помещается в стек (поскольку приоритет левой скобки задан минимальный).
   3. Если входящий элемент является правой скобкой, выгружается стек и добавляются его элементы в очередь, пока не будет встречена левая скобка. После этого левая скобка удаляется из стека.
   4. Если текущий элемент оператор, то проверяется:
      1. Если стек пуст или содержит левую скобку в вершине, то добавляется входящий оператор в стек
      2. Если входящий оператор имеет более высокий приоритет чем вершина, то он помещается в стек.
      3. Если входящий оператор имеет более низкий или равный приоритет, чем вершина, то вершина выгружается в очередь, пока не будет встречен оператор с меньшим приоритетом или левая скобка, затем входящий оператор добавляется в стек.
3. После выполнения алгоритма оставшийся стек выгружается в очередь
4. Очередь записывается справа налево и преобразуется к строке ответа

Вычисление результата выражение по префиксной форме:

1. Инфиксная форма записи преобразуется в list, содержащий введенные числа, операции и скобки
2. Список с выражением в префиксной форме записывается справа налево и последовательно обрабатывается:
   1. Если текущий элемент число, то добавляется в стек.
   2. иначе текущий элемент оператор, то:
      1. Если оператор не найден в списке бинарных операторов, то
         1. Если стек не пуст, то берётся вершина, иначе берётся значение 0 и производится операция, результат выполнения заносится в стек
         2. Переходим к следующему значению из списка выражения в префиксной форме, записанного в обратном порядке, и выполняем алгоритм
      2. В случае, если оператор бинарный, то берутся последние два значения из стека и над ними выполняется операция, результат выполнения заносится в стек.
3. После выполнения алгоритма возвращается последнее значение в стеке

**Пример работы.**

Примеры функциональных тестов положительных сценариев работы программы представлены на рис. 1, 2, 3, 4

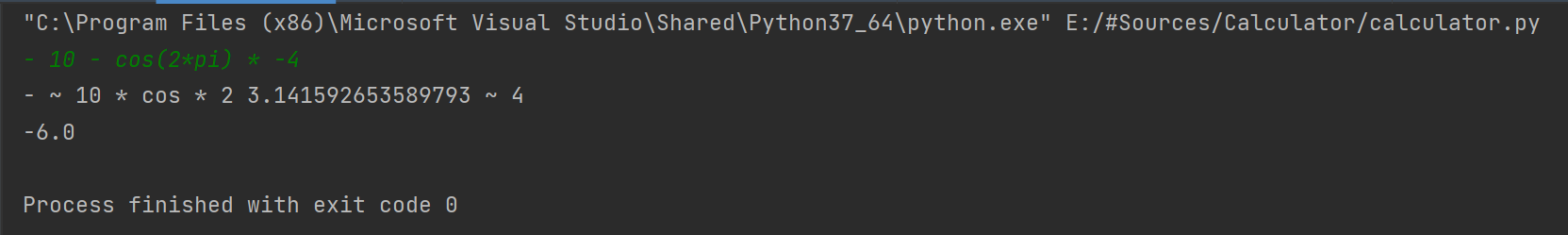


Рис.1 Пример работы программы при вводе выражения - 10 - cos(2\*pi) \* -4

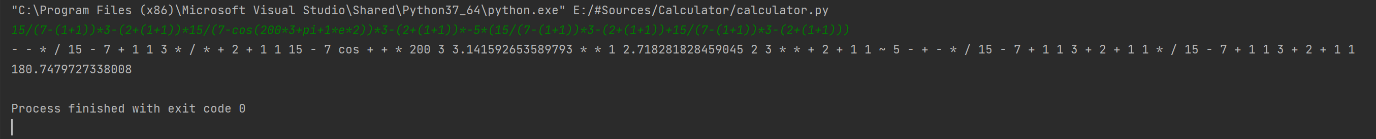


Рис.2 Пример работы программы при вводе выражения 15/(7-(1+1))\*3-(2+(1+1))\*15/(7-cos(200\*3+pi+1\*e\*2))\*3-(2+(1+1))\*-5\*(15/(7-(1+1))\*3-(2+(1+1))+15/(7-(1+1))\*3-(2+(1+1)))

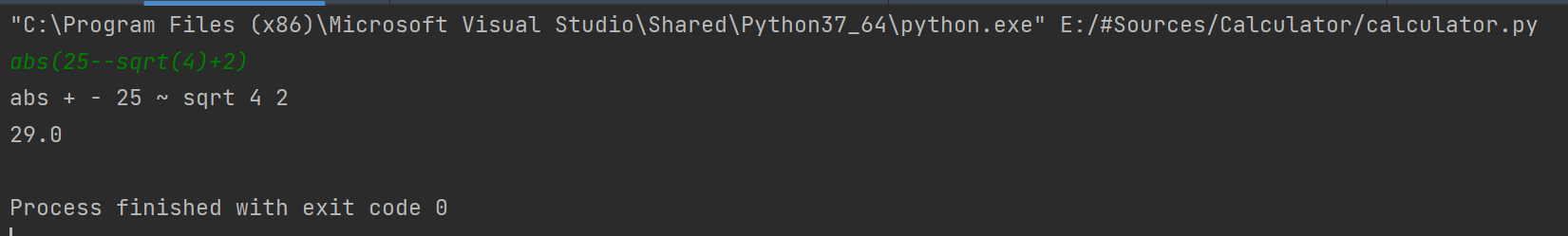


Рис.3 Пример работы программы при вводе выражения abs(25--sqrt(4)+2)

Примеры работы программы при некорректном вводе пользователя представлены на рис. 4, 5, 6

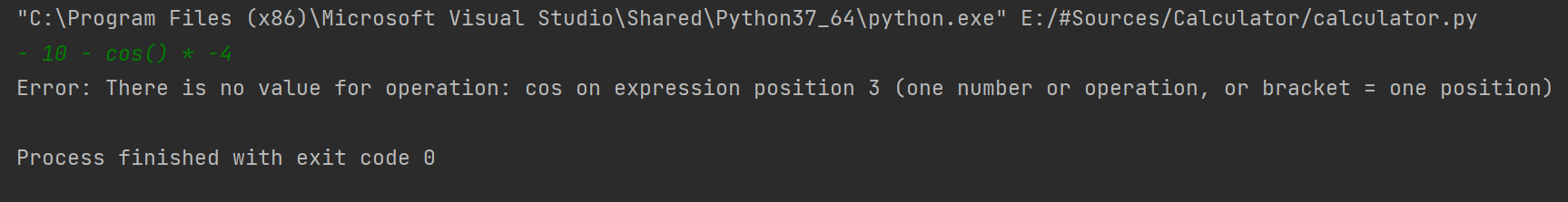


Рис.4 Пример работы программы при вводе выражения - 10 - cos() \* -4

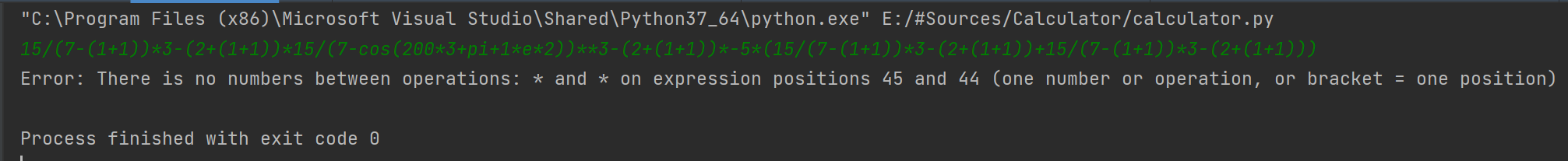


Рис.5 Пример работы программы при вводе выражения 15/(7-(1+1))\*3-(2+(1+1))\*15/(7-cos(200\*3+pi+1\*e\*2))\*3-(2+(1+1))\*-5\*(15/(7-(1+1))\*\*3-(2+(1+1))+15/(7-(1+1))\*3-(2+(1+1)))

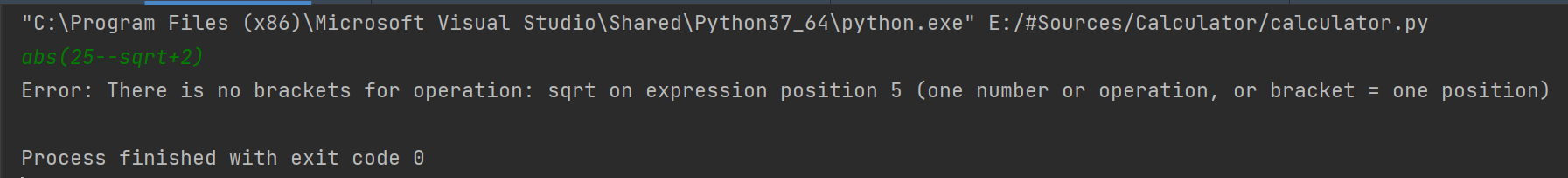


Рис.6 Пример работы программы при вводе выражения abs(25--sqrt+2)

**Листинг.**

calculator.py

import math  
  
  
class ExpressionValidatingException(Exception):  
 pass  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def is\_empty(self):  
 return self.items == []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()  
  
 def peek(self):  
 return self.items[len(self.items) - 1]  
  
 def size(self):  
 return len(self.items)  
  
  
def input\_to\_token(input\_form):  
 *"""Decompose expression string in list   
  
 Args:  
 input\_form: String with expression from user input (infix notation)  
 Returns:  
 list with decomposed expressions  
  
 """* tokenlist = []  
 allowedfunc = ["cos", "sin", "tg", "ctg", "ln", "log", "sqrt", "abs", "pi"]  
 num = ''  
 exp = ''  
 for token in input\_form:  
 if token in "0123456789.":  
 num = num + token  
 elif num != '':  
 tokenlist.append(num)  
 num = ''  
 if token in "cossintgctglnlogsqrtabsp":  
 exp += token  
 elif exp != '':  
 if exp in allowedfunc:  
 tokenlist.append(exp)  
 exp = ''  
 if token in "+-\*/()^e":  
 tokenlist.append(token)  
 if num != '':  
 tokenlist.append(num)  
 if exp != '':  
 tokenlist.append(exp)  
 return tokenlist  
  
  
def check\_input(token\_list):  
 *"""Checks correctness of the entered expression  
  
 Args:  
 token\_list: List with expressions from user input (infix notation)  
  
 Raises:  
 ExpressionValidatingException: if input expression has errors   
  
 """* count = 0  
 for i in range(0, len(token\_list)):  
 if i > 0 and token\_list[i][0] in "0123456789.pe" and token\_list[i - 1][0] in "0123456789.pe)":  
 raise ExpressionValidatingException(  
 "There is no operation between numbers: " + token\_list[i - 1] + " and " + token\_list[  
 i] + " on expression positions " + str(i) + " and " + str(  
 i - 1) + " (one number or operation, or bracket = one position)")  
 if token\_list[i] in "+\*/^" and token\_list[i - 1][0] in "+\*/^":  
 raise ExpressionValidatingException(  
 "There is no numbers between operations: " + token\_list[i - 1] + " and " + token\_list[  
 i] + " on expression positions " + str(i) + " and " + str(  
 i - 1) + " (one number or operation, or bracket = one position)")  
 if token\_list[i] in "cossintgctglnlogsqrtabs":  
 if i == len(token\_list) - 1 or token\_list[i + 1] != "(":  
 raise ExpressionValidatingException(  
 "There is no brackets for operation: " + token\_list[i] + " on expression position " + str(  
 i) + " (one number or operation, or bracket = one position)")  
 hasvalue = False  
 for j in range(i + 1, len(token\_list)):  
 if token\_list[j] == ')':  
 break  
 elif token\_list[j][0] in "0123456789.pe":  
 hasvalue = True  
 break  
 if not hasvalue:  
 raise ExpressionValidatingException(  
 "There is no value for operation: " + token\_list[i] + " on expression position " + str(  
 i) + " (one number or operation, or bracket = one position)")  
 if token\_list[i] == '(':  
 count += 1  
 elif token\_list[i] == ')':  
 count -= 1  
 if count != 0:  
 raise ExpressionValidatingException("Number of ( and ) are not equal!")  
  
  
def backwards(token\_list):  
 *"""Rewrites expression list in reverse order  
  
 Args:  
 token\_list: List with expressions  
 Returns:  
 list with expressions written in reverse order  
  
 """* res = []  
 for i in range(len(token\_list) - 1, -1, -1):  
 if token\_list[i] == '(':  
 res.append(')')  
 elif token\_list[i] == ')':  
 res.append('(')  
 else:  
 res.append(token\_list[i])  
 return res  
  
  
def infix\_to\_prefix(input\_form):  
 *"""Converts infix notation to prefix, also validates input  
  
 Args:  
 input\_form: String with expression from user input (infix notation)  
 Returns:  
 String with expression in prefix form  
  
 """* priority = {"abs": 5, "sqrt": 5, "log": 5, "ln": 5, "ctg": 5, "tg": 5, "sin": 5, "cos": 5, "~": 4, "^": 3, "\*": 2,  
 "/": 2, "+": 1, "-": 1, "(": 0}  
 operation\_stack = Stack()  
 queue = []  
 token\_list = input\_to\_token(input\_form)  
 check\_input(token\_list)  
 token\_list = backwards(token\_list)  
 for i in range(len(token\_list)):  
 if token\_list[i][0] in "0123456789.":  
 queue.append(token\_list[i])  
 elif token\_list[i] in ["pi", "e"]:  
 if token\_list[i] == "pi":  
 queue.append(str(math.pi))  
 else:  
 queue.append(str(math.e))  
 elif token\_list[i] == '(':  
 operation\_stack.push(token\_list[i])  
 elif token\_list[i] == ')':  
 while not operation\_stack.is\_empty() and operation\_stack.peek() != '(':  
 queue.append(operation\_stack.pop())  
 operation\_stack.pop()  
 else:  
 if token\_list[i] == '-' and (  
 i == len(token\_list)-1 or (i < len(token\_list) and token\_list[i + 1] in priority.keys())):  
 token\_list[i] = '~'  
 while (not operation\_stack.is\_empty()) and operation\_stack.peek() != '(' and (  
 priority[operation\_stack.peek()] > priority[token\_list[i]]):  
 queue.append(operation\_stack.pop())  
 operation\_stack.push(token\_list[i])  
  
 while not operation\_stack.is\_empty():  
 queue.append(operation\_stack.pop())  
 queue = backwards(queue)  
 return " ".join(queue)  
  
  
def execute(op, first, second):  
 *"""Execute operation for values.  
  
 Args:  
 op: operation string  
 first: first operand  
 second: second operand  
 Returns:   
 value of executing operation over first and second parameters  
  
 """* if op == "abs":  
 return abs(second)  
 elif op == "sqrt":  
 if second < 0:  
 raise ExpressionValidatingException("Sqrt of negative number sqrt(" + str(second) + ")")  
 return second \*\* 0.5  
 elif op == "log":  
 if second < 0:  
 raise ExpressionValidatingException("logarithm of negative number log(" + str(second) + ")")  
 return math.log10(second)  
 elif op == "ln":  
 if second < 0:  
 raise ExpressionValidatingException("logarithm of negative number ln(" + str(second) + ")")  
 return math.log(second)  
 elif op == "ctg":  
 return math.cos(second) / math.sin(second)  
 elif op == "tg":  
 return math.tan(second)  
 elif op == "sin":  
 return math.sin(second)  
 elif op == "cos":  
 return math.cos(second)  
 elif op == "~":  
 return 0 - second  
 elif op == "^":  
 return first \*\* second  
 elif op == "\*":  
 return first \* second  
 elif op == '/':  
 if second == 0:  
 raise ExpressionValidatingException("Division by zero in operation " + str(first) + " / " + str(second))  
 return first / second  
 elif op == "+":  
 return first + second  
 elif op == "-":  
 return first - second  
  
  
def calc\_prefix(prefix\_form\_list):  
 *"""Calculate value for expression in prefix form.  
  
 Args:  
 prefix\_form\_list: Expression in prefix form decomposed into list  
 Returns:   
 value for expression  
  
 """* operation\_binar = ["^", "\*", "/", "+", "-"]  
 prefix\_form\_list = backwards(prefix\_form\_list)  
 value\_stack = Stack()  
 counter = 0  
 for i in range(len(prefix\_form\_list)):  
 if prefix\_form\_list[i][0] in "0123456789.":  
 value\_stack.push(float(prefix\_form\_list[i]))  
 else:  
 counter += 1  
 if not prefix\_form\_list[i] in operation\_binar:  
 last = value\_stack.pop() if not value\_stack.is\_empty() else 0  
 value\_stack.push(execute(prefix\_form\_list[i], 0, last))  
 continue  
 first = value\_stack.pop() if not value\_stack.is\_empty() else 0  
 second = value\_stack.pop() if not value\_stack.is\_empty() else 0  
 value\_stack.push(execute(prefix\_form\_list[i], first, second))  
 return value\_stack.pop()  
  
  
s=input()  
try:  
 prefixform = infix\_to\_prefix(s)  
 print(prefixform)  
 print(str(calc\_prefix(prefixform.split())))  
except Exception as e:  
 print("Error: " + str(e))  
  
abs(25--sqrt+2)

test\_calculator.py

import calculator  
import pytest  
  
  
def test\_calc\_prefix\_1():  
 prefix\_test = "- 2 ~ ~ 4".split()  
 assert calculator.calc\_prefix(prefix\_test) == -2  
  
  
def test\_calc\_prefix\_2():  
 prefix\_test = "+ - 25 ~ sqrt 4 2".split()  
 assert calculator.calc\_prefix(prefix\_test) == 29.0  
  
  
def test\_calc\_prefix\_3():  
 prefix\_test = "- - \* / 15 - 7 + 1 1 3 \* / \* + 2 + 1 1 15 - 7 + 200 1 3 \* + 2 + 1 1 - + - \* / 15 - 7 + 1 1 3 + 2 + " \  
 "1 1 \* / 15 - 7 + 1 1 3 + 2 + 1 1".split()  
 assert calculator.calc\_prefix(prefix\_test) == -30.072164948453608  
  
  
def test\_infix\_to\_prefix\_1():  
 infix\_form = "25--sqrt(4)+2"  
 assert calculator.infix\_to\_prefix(infix\_form) == "+ - 25 ~ sqrt 4 2"  
  
  
def test\_infix\_to\_prefix\_2():  
 infix\_form = "(1 - 2/3\*pi) \* e / (1/4-5)"  
 assert calculator.infix\_to\_prefix(infix\_form) == "/ \* - 1 \* / 2 3 3.141592653589793 2.718281828459045 - / 1 4 5"  
  
  
def test\_infix\_to\_prefix\_3():  
 infix\_form = "15/(7-(1+1))\*3-(2+(1+1))\*15/(7-cos(200\*3+pi+1\*e\*2))\*3-(2+(1+1))\*-5\*(15/(7-(1+1))\*3-(2+(1+1))+15/(" \  
 "7-(1+1))\*3-(2+(1+1))) "  
 assert calculator.infix\_to\_prefix(  
 infix\_form) == "- - \* / 15 - 7 + 1 1 3 \* / \* + 2 + 1 1 15 - 7 cos + + \* 200 3 3.141592653589793 \* \* 1 " \  
 "2.718281828459045 2 3 \* \* + 2 + 1 1 ~ 5 - + - \* / 15 - 7 + 1 1 3 + 2 + 1 1 \* / 15 - 7 + 1 1 3" \  
 " + 2 + 1 1 "  
  
  
def test\_infix\_to\_prefix\_negative\_1():  
 infix\_form = "(1 - 2/3\*pi)e / (1/4-5)"  
 with pytest.raises(calculator.ExpressionValidatingException):  
 calculator.infix\_to\_prefix(infix\_form)  
  
  
def test\_infix\_to\_prefix\_negative\_2():  
 infix\_form = "(1 - 2/3\*pi)\*e / (1pi \* 4-5)"  
 with pytest.raises(calculator.ExpressionValidatingException):  
 calculator.infix\_to\_prefix(infix\_form)  
  
  
def test\_infix\_to\_prefix\_negative\_3():  
 infix\_form = "(1 - 2/3\*pi)\*e / 1pi \* 4-5)"  
 with pytest.raises(calculator.ExpressionValidatingException):  
 calculator.infix\_to\_prefix(infix\_form)  
  
  
def test\_infix\_to\_prefix\_negative\_4():  
 infix\_form = "(1 - 2/3\*pi)\*e / (1 \* cos() \* 4-5)"  
 with pytest.raises(calculator.ExpressionValidatingException):  
 calculator.infix\_to\_prefix(infix\_form)