ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Курсовой проект

по дисциплине «Функциональное

программирование»

Выполнил студент гр. 3530904/80001: Чижов Н.В.

Руководитель ассистент ВШПИ Лукашин А. А.

Санкт-Петербург

2019

# 1. Задание.

Калькулятор, поддерживающий простые арифметические операции, приоритеты и скобки.

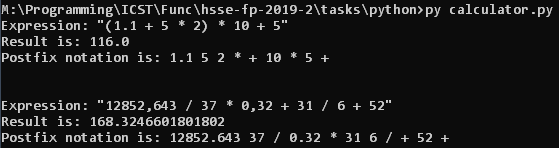
# 2. Ход работы.

## 2.1. Алгоритм решения.

Сначала выражение приводится к стандартному виду: убираются лишние пробелы/табуляции/переносы строки, заменяются запятые на точки. Приведенное выражение передается в функцию, переводящую выражение из инфиксной формы в постфиксную. Процесс выполняется при помощи стека.

Выражение в постфиксной форме переходит уже в функцию расчета. Выражение читается слева направо: постоянные помещаются в стек, выражения применяются к двум верхним постоянным в стеке.

## 2.2. Результат работы.



# 3. Выводы.

В ходе работы был изучен функциональный подход к программированию в языке программирования Python 3, который значительно

отличается от стандартного императивного подхода. Изучены некоторые основные алгоритмы,

используемые в функциональном программировании и произведена работа с ними.

# Приложение.

## calculator.py

1. **def** readOperator(expr: str):
2. isNumber = expr[0].isdigit()
3. i = 0
4. **if** (isNumber):
5. **point = False**
6. **for** c **in** expr:
7. **if** (c.isdigit()):
8. i += 1
9. **elif** (c == '.'):
10. **if point:**
11. **raise** 2
12. i += 1
13. point = True
14. **else**:
15. **break**
16. **else**:
17. **if** (expr[0] **not** **in** ['+', '-', '\*', '/', '(', ')']):
18. **raise** 2;
19. i = 1
21. **return** (expr[:i], expr[i:])
23. **def** isFloat(num: str):
24. **try**:
25. **float(num)**
26. **return** True
27. **except** ValueError:
28. **return** False
30. **sanitize = lambda expr: "".join(expr.split()).replace(',', '.')**
32. **def** infixToPostfix(expr: str):
33. expr = sanitize(expr)
34. high = ['\*', '/']
35. **low = ['+', '-']**
37. stack = []
38. opStack = []
39. **while** (len(expr)):
40. **op, expr = readOperator(expr)**
42. **if** (op.isdigit() **or** isFloat(op)):
43. stack += [op]
44. **elif** (op == '('):
45. **opStack += [op]**
46. **elif** (op == ')'):
47. **while** (1):
48. **if** (opStack[-1] == '('):
49. opStack = opStack[:-1]
50. **break**
51. **else**:
52. stack, opStack = stack + [opStack[-1]], opStack[:-1]
53. **else**:
54. **while** (1):
55. **if ((not len(opStack)) or opStack[-1] == '('):**
56. opStack += [op]
57. **break**
58. **else**:
59. **if** (opStack[-1] **in** low) **and** (op **in** high):
60. **opStack += [op]**
61. **break**
63. **if** ((opStack[-1] **in** high) **and** (op **in** high)) **or** ((opStack[-1] **in** low) **and** (op **in** low)):
64. stack, opStack = stack + [opStack[-1]], opStack[:-1] + [op]
65. **break**
67. **if** (opStack[-1] **in** high) **and** (op **in** low):
68. stack, opStack = stack + [opStack[-1]], opStack[:-1]
69. **else**:
70. **break**
72. stack += opStack[::-1]
73. **return** stack
75. **def calculate(expr: str):**
76. expr = sanitize(expr)
77. ops = ['+', '-', '\*', '/']
78. postfix = infixToPostfix(expr)
79. stack = []
80. **for op in postfix:**
81. **if** (op **in** ops):
82. rhs, lhs, stack = stack[-2], stack[-1], stack[:-2]
83. stack += [eval('%s %s %s' % (rhs, op, lhs))]
84. **else**:
85. **stack += [op]**
87. **return** stack[0]
89. **def** outputExpression(expr: str):
90. **print("Expression: \"%s\"" % expr)**
91. **print**("Result is: %s" % calculate(expr))
92. **print**("Postfix notation is: %s" % ' '.join(infixToPostfix(expr)))
94. **def** main():
95. **outputExpression('(1.1 + 5 \* 2) \* 10 + 5')**
96. **print**('**\n**')
97. outputExpression('12852,643 / 37 \* 0,32 + 31 / 6 + 52') *# Thank you, @Polykek2K!*
99. **if \_\_name\_\_ == "\_\_main\_\_":**
100. main()