# 5.3. Листинги

|  |
| --- |
| Калькулятор в двух файлах с интерфейсом  файл с функциями calculator\_logic.py  def add(a, b):  return a + b  def subtract(a, b):  return a - b  def multiply(a, b):  return a \* b  def divide(a, b):  if b != 0:  return a / b  else:  return "Error: Division by zero" |
| from tkinter import \*  from tkinter import ttk  import calculator\_logic as c  def calc():  second = float(entry.get())  if oper == '+':  result = c.add(first, second)  elif oper == '-':  result = c.subtract(first, second)  elif oper == '\*':  result = c.multiply(first, second)  elif oper == '/':  result = c.divide(first, second)  entry.delete(0, END)  entry.insert(0, str(result))  def enter\_number(number):  entry.insert(END, number)  def set\_operation(operation):  global first  global oper  first = float(entry.get())  oper = operation  entry.delete(0, END)  def clear\_entry():  entry.delete(0, END)  # Создаем окно приложения  window = Tk()  window.title("Калькулятор")  entry = ttk.Entry(width=20)  entry.grid(row=0, column=0, columnspan=4, sticky="ew")  # Цифровые кнопки  ttk.Button(text="1", command=lambda: enter\_number('1')).grid(row=1, column=0)  ttk.Button(text="2", command=lambda: enter\_number('2')).grid(row=1, column=1)  ttk.Button(text="3", command=lambda: enter\_number('3')).grid(row=1, column=2)  ttk.Button(text="4", command=lambda: enter\_number('4')).grid(row=2, column=0)  ttk.Button(text="5", command=lambda: enter\_number('5')).grid(row=2, column=1)  ttk.Button(text="6", command=lambda: enter\_number('6')).grid(row=2, column=2)  ttk.Button(text="7", command=lambda: enter\_number('7')).grid(row=3, column=0)  ttk.Button(text="8", command=lambda: enter\_number('8')).grid(row=3, column=1)  ttk.Button(text="9", command=lambda: enter\_number('9')).grid(row=3, column=2)  ttk.Button(text="0", command=lambda: enter\_number('0')).grid(row=4, column=0)  # Кнопки арифметических операций  ttk.Button(text="+", command=lambda: set\_operation('+')).grid(row=1, column=3)  ttk.Button(text="-", command=lambda: set\_operation('-')).grid(row=2, column=3)  ttk.Button(text="\*", command=lambda: set\_operation('\*')).grid(row=3, column=3)  ttk.Button(text="/", command=lambda: set\_operation('/')).grid(row=4, column=3)  # Кнопка равно и очистки  ttk.Button(text="=", command=calc).grid(row=4, column=2)  ttk.Button(text="C", command=clear\_entry).grid(row=4, column=1)  first = 0  oper = 0  window.mainloop() |
| Добавляем проверку поля ввода с использованием генератора списков  import tkinter as tk  from tkinter import ttk  import calculator\_logic as calc  def validate\_entry():  entry\_text = entry.get()  corrected\_text = ''.join(b for b in entry\_text if b in "0123456789.-")  if entry\_text != corrected\_text:  entry.delete(0, END)  entry.insert(0, corrected\_text)  def enter\_number(number):  entry.insert(END, number)  def set\_operation(operation):  global first  global oper  try:  first = float(entry.get())  except ValueError:  first = 0  oper = operation  entry.delete(0, END)  def calculate\_result():  second = float(entry.get())  if oper == '+':  result = calc.add(first, second)  elif oper == '-':  result = calc.subtract(first, second)  elif oper == '\*':  result = calc.multiply(first, second)  elif oper == '/':  result = calc.divide(first, second)  entry.delete(0, END)  entry.insert(0, str(result))  def clear\_entry():  entry.delete(0, END)  # Создаем окно приложения  window = Tk()  window.title("Calculator")  entry = ttk.Entry(width=20)  entry.grid(row=0, column=0, columnspan=3)  entry.bind('<KeyRelease>', lambda event: validate\_entry())  # Цифровые кнопки  ttk.Button(text="1", command=lambda: enter\_number('1')).grid(row=1, column=0)  ttk.Button(text="2", command=lambda: enter\_number('2')).grid(row=1, column=1)  ttk.Button(text="3", command=lambda: enter\_number('3')).grid(row=1, column=2)  ttk.Button(text="4", command=lambda: enter\_number('4')).grid(row=2, column=0)  ttk.Button(text="5", command=lambda: enter\_number('5')).grid(row=2, column=1)  ttk.Button(text="6", command=lambda: enter\_number('6')).grid(row=2, column=2)  ttk.Button(text="7", command=lambda: enter\_number('7')).grid(row=3, column=0)  ttk.Button(text="8", command=lambda: enter\_number('8')).grid(row=3, column=1)  ttk.Button(text="9", command=lambda: enter\_number('9')).grid(row=3, column=2)  ttk.Button(text="0", command=lambda: enter\_number('0')).grid(row=4, column=0)  # Кнопки арифметических операций  ttk.Button(text="+", command=lambda: set\_operation('+')).grid(row=1, column=3)  ttk.Button(text="-", command=lambda: set\_operation('-')).grid(row=2, column=3)  ttk.Button(text="\*", command=lambda: set\_operation('\*')).grid(row=3, column=3)  ttk.Button(text="/", command=lambda: set\_operation('/')).grid(row=4, column=3)  # Кнопка равно и очистки  ttk.Button(text="=", command=calculate\_result).grid(row=4, column=2)  ttk.Button(text="C", command=clear\_entry).grid(row=4, column=1)  first = 0  oper = 0  window.mainloop() |
| функция `validate\_entry` проверяет, содержит ли поле ввода только допустимые символы (цифры, точки и минусы) и корректирует содержимое, если это не так.  1. \*\*Получение текста из поля ввода:\*\*  entry\_text **=** entry.get()  2. \*\*Фильтрация допустимых символов:\*\*  corrected\_text **=** ''.join(b **for** b **in** entry\_text **if** b **in** "0123456789.-")  Здесь создается новая строка `corrected\_text`, которая состоит только из символов из `entry\_text`, принадлежащих набору "0123456789.-". Это достигается с помощью генератора списков, который проходит по каждому символу `b` в `entry\_text` и включает его в результат, если `b` содержится в строке "0123456789.-".  3. \*\*Сравнение и коррекция текста:\*\*  **if** entry\_text **!=** corrected\_text:  entry.delete(0, END)  entry.insert(0, corrected\_text)  Этот блок сравнивает исходный текст `entry\_text` с отфильтрованным `corrected\_text`. Если они не совпадают, это означает, что в исходном тексте были символы, которые не входят в допустимый набор. В этом случае:  - `entry.delete(0, END)` удаляет весь текущий текст из поля ввода.  - `entry.insert(0, corrected\_text)` вставляет отфильтрованный текст обратно в поле ввода. |
| `corrected\_text = ''.join(b for b in entry\_text if b in "0123456789.-")`  использует генератор списков в сочетании с методом `join` для создания строки, содержащей только определенные символы из исходной строки `entry\_text`. Разберем эту строку пошагово:  \*\*Генератор списков:\*\*  (b **for** b **in** entry\_text **if** b **in** "0123456789.-")  Это генератор списков, который итерируется по каждому символу `b` в строке `entry\_text`. Для каждого символа `b` он проверяет, содержится ли этот символ в строке `"0123456789.-"`, которая представляет собой набор допустимых символов (цифры, точку и минус).  3. \*\*Условие фильтрации:\*\*  **if** b **in** "0123456789.-"  Это условие в генераторе списков. Оно означает, что в результирующий список (а в данном случае — в строку) будут включены только те символы `b`, которые присутствуют в строке `"0123456789.-"`. Если символ `b` не содержится в этом наборе, он игнорируется и не добавляется в `corrected\_text`.  4. \*\*Метод `join`:\*\*  ''.join(**...**)  Метод `join` объединяет элементы итерируемого объекта (в данном случае, генератора списков) в одну строку. В данном случае, перед `join` стоит пустая строка `''`, что означает, что символы будут соединены без каких-либо разделителей.  Таким образом, `corrected\_text` становится строкой, содержащей только те символы из `entry\_text`, которые допустимы (цифры, точки и минусы). Например, если `entry\_text` был `"abc123.45-xyz"`, то `corrected\_text` будет `"123.45-"`, так как только эти символы присутствуют в допустимом наборе `"0123456789.-"`. |
| Отладка |
|  |
| Юнит тесты создаем файл test\_calculator\_logic.py  import calculator\_logic as calc  def test\_add():  assert calc.add(10, 5) == 15  assert calc.add(-1, 1) == 0  assert calc.add(-1, -1) == -2  def test\_subtract():  assert calc.subtract(10, 5) == 5  assert calc.subtract(-1, 1) == -2  assert calc.subtract(-1, -1) == 0  def test\_multiply():  assert calc.multiply(10, 5) == 50  assert calc.multiply(-1, 1) == -1  assert calc.multiply(-1, -1) == 1  def test\_divide():  assert calc.divide(10, 5) == 2  # Проверка исключения при делении на ноль  try:  calc.divide(10, 0)  assert False # Эта строка не должна выполняться, если исключение произошло  except ValueError:  assert True # Если исключение произошло, тест пройден  assert calc.divide(-1, 1) == -1  assert calc.divide(-1, -1) == 1  test\_add()  test\_subtract()  test\_multiply()  test\_divide()  print("Все тесты пройдены успешно!") |
| def test\_divide():  assert c.divide(10, 2) == 5  assert c.divide(-10, 2) == -5  assert c.divide(10, -2) == -5 |
| Блок try-except:  В блоке try выполняется потенциально опасная операция, которая может вызвать исключение. В данном случае, это calc.divide(10, 0). Ожидается, что при делении на ноль будет сгенерировано исключение ValueError.  assert False:  Ключевое слово assert используется для проверки утверждений. Если условие после assert истинно (True), тест продолжается. Если ложно (False), тест вызывает исключение AssertionError и останавливается. В этом случае, если строка assert False выполняется, это означает, что исключение ValueError не было сгенерировано (что является ошибкой в логике теста).  Обработка исключения ValueError:  Если при выполнении кода в блоке try возникает исключение ValueError, управление передается в блок except. Здесь assert True выполняется, что означает, что тест прошел успешно (поскольку исключение ValueError было успешно обнаружено и обработано).  Итоговый результат:  Если деление на ноль в calc.divide(10, 0) вызывает ValueError, как и ожидалось, тест проходит успешно. Если же исключение не возникает, это означает ошибку в логике функции divide, и тест не будет пройден, сгенерировав AssertionError.  Таким образом, использование assert в юнит-тестах позволяет проверить, соответствует ли поведение программы ожидаемому. Это важный инструмент для обеспечения корректности и надежности кода. |
| Переделываем функцию деления чтобы она выдавала исключения  def divide(a, b):  if b == 0:  raise ValueError("Division by zero")  return a / b |
| import calculator\_logic as c  import pytest  def test\_add():  assert c.add(10, 5) == 15  assert c.add(-1, 1) == 0  assert c.add(-2, -2) == -4  with pytest.raises(TypeError):  c.add("text", 1)  with pytest.raises(TypeError):  c.add(1, "text")  def test\_subtract():  assert c.subtruct(10, 5) == 5  assert c.subtruct(-1, 1) == -2  assert c.subtruct(-10, -10) == 0  with pytest.raises(TypeError):  c.subtruct("text", 1)  with pytest.raises(TypeError):  c.subtruct(1, "text")  def test\_multiply():  assert c.multiply(2, 2) == 4  assert c.multiply(-5, 6) == -30  assert c.multiply(-1, -5) == 5  with pytest.raises(TypeError):  c.multiply("text", 1)  with pytest.raises(TypeError):  c.multiply(1, "text")  def test\_divide():  assert c.divide(10, 2) == 5  assert c.divide(-10, 2) == -5  assert c.divide(10, -2) == -5  with pytest.raises(TypeError):  c.divide("text", 1)  with pytest.raises(TypeError):  c.divide(1, "text")  with pytest.raises(ZeroDivisionError):  c.divide(10, 0)  test\_add()  test\_subtract()  test\_multiply()  test\_divide()  print("Все тесты пройдены успешно!") |