Calculator Documentation

Release 0.1

Muhammad Yaseen
Geetha Rajadurai
Ali Khozravi
Majid Zeraati
Mortaza Jalalvand

Mar 15, 2017

CONTENTS:

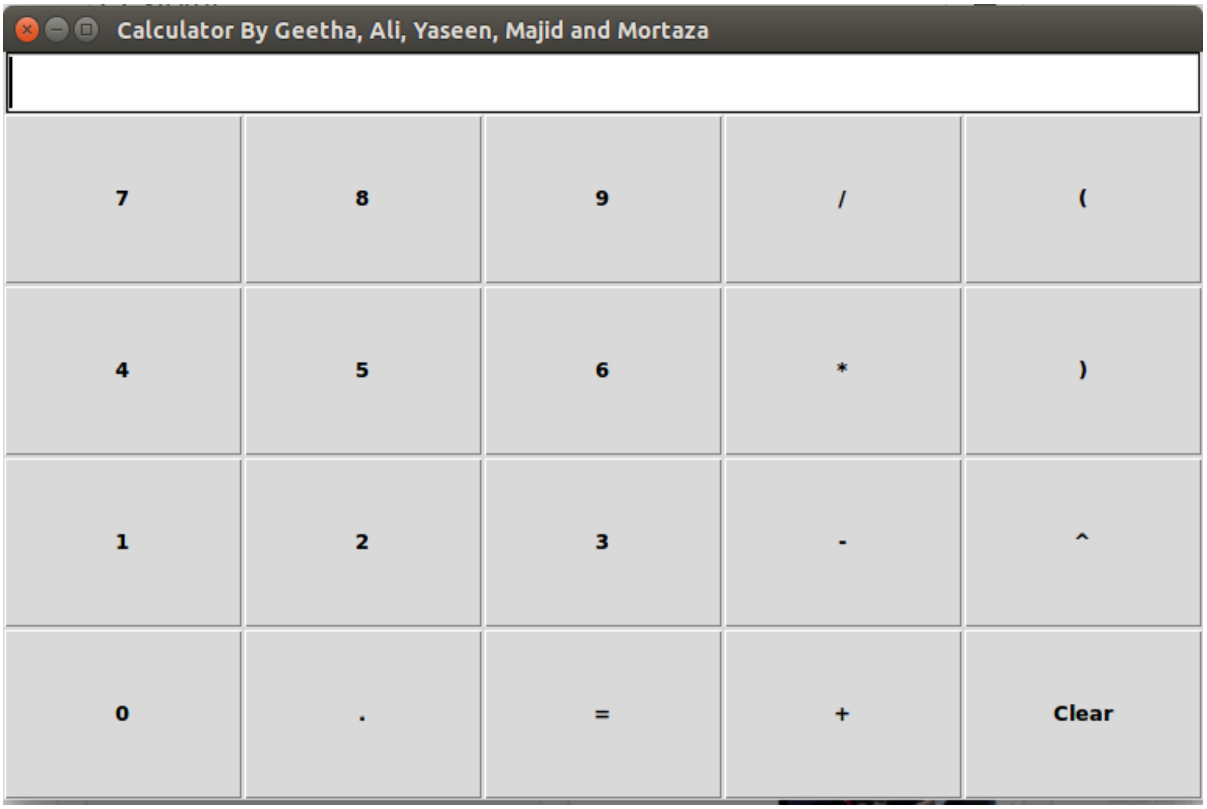

ICTP
403
M+ M- Mc ÷
7 8 9 x
4 5 6 −
1 2 3 +
0 . =
ICTP

CHAPTER

ONE

INTRODUCTION

1.1 GUI Class in Python



Calculator is a `Python` GUI to visualise the C++ results. It can solve basic mathematical expressions containing `+`, `-`, `*`, `/`, `**`, and `()`. The development version of the package is available on Github.

1.1.1 Python Code for GUI Calculator

```
# By Geetha, Ali, Yaseen, Majid and Mortaza
# import Tkinter as Tk # Python2
import tkinter as Tk # Python3
import subprocess
```

```
class Calculator:
    # Constructor for adding buttons
    def __init__(self, window):
        window.title('Calculator By Geetha, Ali, Yaseen, Majid and Mortaza')
        window.geometry()
        self.text_box = Tk.Entry(window, width=40, font="Noto 20 bold")
        self.text_box.grid(row=0, column=0, columnspan=6)
        self.text_box.focus_set()
        # Buttons
        Tk.Button(window,text="+",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('+')).grid(row=4, column=3)
        Tk.Button(window,text="*",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('*')).grid(row=2, column=3)
        Tk.Button(window,text="-",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('-')).grid(row=3, column=3)
        Tk.Button(window,text="/",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('/')).grid(row=1, column=3)
        Tk.Button(window,text="7",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('7')).grid(row=1, column=0)
        Tk.Button(window,text="8",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(8)).grid(row=1, column=1)
        Tk.Button(window,text="9",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(9)).grid(row=1, column=2)
        Tk.Button(window,text="4",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(4)).grid(row=2, column=0)
        Tk.Button(window,text="5",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(5)).grid(row=2, column=1)
        Tk.Button(window,text="6",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(6)).grid(row=2, column=2)
        Tk.Button(window,text="1",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(1)).grid(row=3, column=0)
        Tk.Button(window,text="2",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(2)).grid(row=3, column=1)
        Tk.Button(window,text="3",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(3)).grid(row=3, column=2)
        Tk.Button(window,text="0",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(0)).grid(row=4, column=0)
        Tk.Button(window,text=".",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('.')).grid(row=4, column=1)
        Tk.Button(window,text="(",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('(')).grid(row=1, column=4)
        Tk.Button(window,text=")",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action(')')).grid(row=2, column=4)
        Tk.Button(window,text="=",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.equals()).grid(row=4, column=2)
        Tk.Button(window,text="^",font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.action('^')).grid(row=3, column=4)
        Tk.Button(window,text='Clear',font="Noto 10 bold",width=14,height=6,
↪command=lambda:self.clearall()).grid(row=4, column=4)

    def action(self, arg):
        """Attaching button's value to end of the text box"""
        self.text_box.insert(Tk.END, arg)

    def get(self):
        """Getting expression from c++ code"""
        self.expression = self.text_box.get()
```

```
    def equals(self):
        self.get()
        self.expression=self.expression.replace('(','\(') # Because of echo!
        self.expression=self.expression.replace(')','\)') # Because of echo!
        self.value= subprocess.check_output("echo {} | ./main.x".format(self.
↪expression), shell=True)
        self.text_box.delete(0, Tk.END)
        self.text_box.insert(0, self.value)

    def clearall(self):
        """Clearing the text box"""
        self.text_box.delete(0, Tk.END)


window = Tk.Tk()
ob = Calculator(window)
window.mainloop()
```

1.2 Calculator in C++

Calculator is a C++ progamm to solve basic mathematical expressions. It can solve basic mathematical expressions containing +, −, ⋆, /, ⋆⋆, and (). The development version of the package is available on Github.

1.2.1 Code for Main Function

```
#include <iostream>
#include "tokenizer.h"
#include "parser.h"

using std::cout;
using std::endl;
using namespace std;

int main () {
  Tokenizer tokenizer;
  Parser parser;
  std::string line;
  while (!std::cin.eof()) {
    std::getline (std::cin, line);
    if (!line.empty())
    {
      std::vector<Token> res = tokenizer.split(line);
      std::cout << "result: " << parser.parse_line(res);
    }
  }
  return 0;
}
```

1.2.2 Code for Tokenizer Function

```
#include "tokenizer.h"
#include <iostream>
```

```
#include <cstdlib>
#include "error.h"

std::vector<Token> Tokenizer::split (std::string str)
{
  std::vector<Token> result;
  for (int i=0; i<str.length(); ++i) {
    char c = str[i];
    if (c=='+')
    {
      std::string op;
      op += c;
      result.push_back(Token(PLUS,op));
    }
    else if (c=='-')
    {
      std::string op;
      op += c;
      result.push_back(Token(MINUS,op));
    }
    else if (c=='*')
    {
      std::string op;
      if (str[i+1]=='*')
      {
        op+= '^';
        result.push_back(Token(POW,op));
        i++;
      }
      else
      {
        op+= c;
        result.push_back(Token(STAR,op));
      }
    }
    else if (c=='^')
    {
      std::string op;
      op += c;
      result.push_back(Token(POW,op));
    }
    else if (c=='/')
    {
      std::string op;
      op += c;
      result.push_back(Token(SLASH,op));
    }
    else if (c=='(')
    {
      std::string op;
      op += c;
      result.push_back(Token(OPENPTS,op));
    }
    else if (c==')')
    {
      std::string op;
      op += c;
      result.push_back(Token(CLOSEPTS,op));
```

```
    }
    else if  (c=='.')
    {
      std::string number;
      number += str[i];
      i++;
      while(isdigit(str[i]))  number+=str[i++];
      if (str[i]=='.')
      {
        error_exit ("Invalid float!");
      }
      result.push_back(Token(NUMERIC,number));
      i--;
    }
    else if (isblank(c)) continue;
    else if (isdigit(c))
    {
      int dot_counter=0;
      std::string number;
      while(isdigit(str[i]) || (str[i]=='.'))
      {
        if (str[i]=='.') dot_counter++;
        number+=str[i++];
      }
      if (dot_counter>1)
      {
        error_exit ("Invalid float!");
      }
      result.push_back(Token(NUMERIC,number));
      i--;
    }
    else
      error_exit ("Unknown character");
  }
  return result;
}
```

1.2.3 Code for Parser Function

```
#include "tokenizer.h"
#include "parser.h"
#include <cmath>
#include "error.h"

double Parser::parse_line (const std::vector<Token> &t) {
  tokens = t;
  counter = 0;
  return expression ();
}

double Parser::factor ()
{
  double result = primary ();
  if (counter + 1 < tokens.size() && tokens[counter + 1].kind == POW)
  {
```

```
    counter += 2;
    if (counter >= tokens.size())
      error_exit("Syntax of power function incorrect!");
    double exponent = primary();
    return std::pow (result, exponent);
  }
  else
    return result;
}

double Parser::primary ()
{
  if (tokens[counter].kind == NUMERIC)
  {
    return stod (tokens[counter].val);
  }
  else if (tokens[counter].kind == OPENPTS)
  {
    counter++;
    return expression();
  }
  else if (tokens[counter].kind == MINUS)
  {
    counter++;
    if (tokens[counter].kind == OPENPTS)
    {
      counter++;
      return -expression();
    }
    else
      return -stod (tokens[counter].val);
  }
  else if (tokens[counter].kind == PLUS)
  {
    counter++;
    if (tokens[counter].kind == OPENPTS)
    {
      counter++;
      return +expression();
    }
    else
      return +stod (tokens[counter].val);
  }
  else
    return 0; // error
}

double Parser::term ()
{
  double result = factor();
  counter++;
  while (counter<tokens.size()&&(tokens[counter].kind == STAR || tokens[counter].kind␣
↪== SLASH))
  {
    if (tokens[counter].kind == STAR)
    {
      counter++;
      result *= factor();
```

```
      counter++;
    }
    else
    {
      counter++;
      result /= factor();
      counter++;
    }
  }
  return result;
}

double Parser::expression ()
{
  double result = term();
  while (counter<tokens.size()&&(tokens[counter].kind == PLUS || tokens[counter].kind
↪== MINUS))
  {
    if (tokens[counter].kind == PLUS)
    {
      counter++;
      result += term();
    }
    else
    {
      counter++;
      result -= term();
    }
  }
  return result;
}
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search