Implementing "Shunting Yard" algorithm

AMS 595

Fundamentals of Computing, Fall 2020

Instructor: Jarret Petrillo, Oliver Yang

Jenny Chen(111251403)

Nov. 29.2020

Objective

Implementing "Shunting Yard" algorithm to convert an infix expression into a postfix expression and evaluate the result in the calculator using Python. The shunting yard algorithm apply a scientific calculator that support floating point numbers and trigonometric function. Design interface of calculator style by using a module- tkinter from Python and import modules - Calculator from my class into the main module.

First, setting up twenty-four buttons and each button associate with the input as a dictionary that processed one symbol at a time: if a number is found, it is copied directly to the output. If the symbol is an operator, it is pushed onto the operator stack. If the operator's precedence is lower than that of the operators at the top of the stack or the precedents are equal and the operator is left associative, then that operator is popped off the stack and added to the output. If the left parenthesis, push it on the stack; and if the incoming symbol is a right parenthesis, pop and print the stack symbols until you see a left parenthesis. Finally, any remaining operators are popped off the stack and added to the output. I used stack to hold operators rather than numbers. The purpose of the stack is to reverse the order of the operators in the expression.

Methods and Tools

1. Import the module -Calculator by adding

```
if __name__ == "__main__":
calculator = Calculator()
calculator.createUI()
```

   in the end to run the code.

2. Class Calculator define as a module that contains executable functions and the code intended to initialize the module as well:

   Import module tkinter from Python and design the Calculator interface style as a function:

```
class Calculator:
    def __init__(self):
        #GUI elements
        self.formulaLbl = None
        self.resultLbl = None
        self.sinBtn = None
        self.cosBtn = None
        self.tanBtn = None
        self.backBtn = None
        self.leftParentheseBtn = None
        self.rightParentheseBtn = None
        self.clearBtn = None
        self.divideBtn = None
        self.sevenBtn = None
        self.eightBtn = None
        self.NineBtn = None
        self.minusBtn = None
        self.fourBtn = None
        self.fiveBtn = None
        self.sixBtn = None
        self.minusBtn = None
        self.oneBtn = None
        self.twoBtn = None
        self.threeBtn = None
        self.plusBtn = None
        self.negBtn = None
        self.zeroBtn = None
        self.pointBtn = None
        self.equalBtn = None
```

```
def createUI(self):
       self.sevenBtn = tk.Button(root, text="7", font=mediumFont)
       self.sevenBtn.grid(row=5, column=0, padx=5, pady=5, sticky="NESW")
       self.sevenBtn.bind('<Button-1>', self.sevenBtnHandler)
```

   Create each bottom as a dictionary such as :

```
self.input.append({
"is_operator": False ( print),
"display": "button",
"operant_count": number,
```

"precedence": order number,

"associativity": **""**,

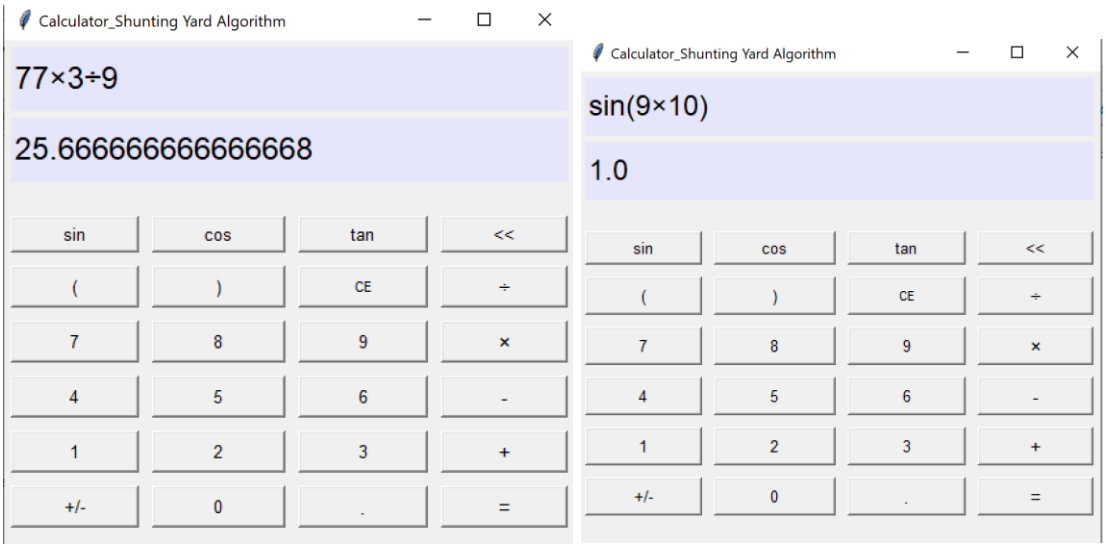
"is_function": **False (numbers or operator)**,

"operator": operator})

3. Implement the function value.

- If **"is_operator"**: False, than display it**.**
- If **"operant_count":** 0 is parenthesis and 1 is number. If the left parenthesis, push it on the stack. And if the incoming symbol is a right parenthesis, pop and print the stack symbols until you see a left parenthesis. Pop the left parenthesis and discard it.
- If **"is_operator"**: True, than **"precedence" order:** If the incoming symbol is an operator and has either higher precedence than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is right associative -- push it on the stack. If the incoming symbol is an operator and has either lower precedence than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is left associative -- continue to pop the stack until this is not true. Then, push the incoming operator.
- If **"is_function":** False is numbers and operators, otherwise is false**.**
- Finally, display the expression, pop and print all operators on the stack. (No parentheses should remain.)

Result and experience



1. Identification of basis:

   In Python, tkinter toolkits is a fantastic way to create graphical desktop applications such as display calculator result by implement shunting yard algorithm.

   Calculator as a class that contains many functions and encapsulate as a module that can be import into the main module.

2. Accomplishment:

   Design a calculator for support floating point numbers and trigonometric function

   by implementing "Shunting Yard" algorithm. Python has a great module – tkinter to design GUI. Since there are twenty-four buttons involve so that a longer program is better off set input for the interpreter and run it as a module. A module can contain executable statements that initialize the model with import the module.

3. Results:

   Python has a great toolkit to design a scientific calculator for the desktop interface. Import module to support floating point numbers and trigonometric function into the main module.

Conclusions

Use module to split a long program make the code easy to maintenance, reusing and testing. By import module into the main module that I can access to a script executed at the top level.

Reference:

https://en.wikipedia.org/wiki/Shunting-yard_algorithm