

Objective: "Implement a program that will take ANY such formula for the Caesar Cipher. You will use/need this for the first Midterm exam. Use the $3+P \bmod 26$ initially for your assignment; make sure your program can handle ANY formula."

What Worked: This was actually pretty easy. It was merely using the Shunting Yard Algorithm to convert our input into postfix for our cypher.

What Didn't: The hard part (in just realization) was that while parenthesis are given highest priority in our order of operations, we have to give it the lowest priority because it's not actually an operator. Additionally, I forgot that *mod* has the same priority as multiplication and division. Thusly it took me a few minutes to figure out that $P + 3 \bmod 26$ should be represented as $(P + 3) \bmod 26$ to correctly mod by the length of the alphabet.

Comments: It allowed me to re-visit reverse polish notation. This was the first assignment given to me in *Algorithms and Data Structures* it is also something that I glossed over and didn't appreciate the first time through. I found that this youtube video [<http://youtu.be/QzVVjboyb0s?t=9m25s>] has a very nice run through of the algorithm.

```
import re

Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Operators = { # Setup Operators and their priorities. P-Lease E-xcuse M-y D-ear A-unt S-ally
    "+" : 0,
    "-" : 0,
    "*" : 1,
    "/" : 1,
    "%" : 1,
    "^" : 2,
    "(" : -1, # Turns out we definitely need paren here for comparisons against operators. HOWEVER they must have a lower priority
    ")" : -1 # to allow for checking the stack order correctly. This is because Paren are not actually operators BUT we have to check against them in our stack. This does not cause any problems because we already have a separate case for handling paren, even though they both use the same stack.
}

def InfixToPostfix(EquationString):
    EquationString = EquationString.upper()
    EquationString = re.sub("(MOD)", "%", EquationString)
    EquationString = re.sub("(\\s+)", "", EquationString) # removing unnecessary spaces.
    Q = []
    Stk = []
    # ^(\d+) #Number at begging of string regex
    # ^([+\\*-\\/\\^%]) #Operator at beggining of string regex
    # ^([\\(\\)]) #matchs oppening and closing paren.
```

```

while len(EquationString) > 0:
    DigitSearch = re.search(r"^(\d+|P)", EquationString) # tests for digit, OR variable
    N
    if (DigitSearch):
        EquationString = re.sub(r"^(\d+|P)", "", EquationString)
        if DigitSearch.group(1) == "P":
            Q.append(DigitSearch.group(1))
        else:
            Q.append(int(DigitSearch.group(1)))
        continue
    OperatorSearch = re.search(r"^([\+\*\-\\/\^%])", EquationString)
    if (OperatorSearch):
        EquationString = re.sub(r"^([\+\*\-\\/\^%])", "", EquationString)
        op = OperatorSearch.group(1)
        while (len(Stk) > 0 and Operators[Stk[len(Stk) - 1]] >= Operators[op]):
            Q.append(Stk.pop())
        Stk.append(op)
        continue
    ParenSearch = re.search(r"^([\(\)])", EquationString)
    if (ParenSearch):
        EquationString = re.sub(r"^([\(\)])", "", EquationString)
        if (ParenSearch.group(1) == "("):
            Stk.append(ParenSearch.group(1))
        else: # right paren
            while not Stk[len(Stk) - 1] == "(":
                Q.append(Stk.pop())
            Stk.pop() # ditch the left paren
        continue
    #if you want to implement additional mathematical functionality, this is the spot to
    #start including it. Otherwise, this is an invalid infix equation.
    raise Exception("Not a valid infix equation. {0}".format(EquationString)) # If we
    #reach this line, something has probably gone wrong.
    http://www.youtube.com/watch?v=Q5cEAhjcv54
    while len(Stk) > 0:
        Q.append(Stk.pop())
    return Q

def ParseRPN(RpnArr, PValue):
    RpnArr = list(RpnArr) # we need a deep copy. If we don't we'll override P and it'll
    #stay that way through the duration of our encryption.
    ParseStk = []
    for i in range(0, len(RpnArr)):
        if (RpnArr[i] == "P"):
            RpnArr[i] = PValue # make the mathematical substitution here so we can
            #correctly evaluate this equation.
        if (isinstance(RpnArr[i], int)):
            ParseStk.append(RpnArr[i])

```

```

    else:
        A = ParseStk.pop()
        B = ParseStk.pop()
        # I have recently researched a more Pythonic way of doing this,
        # however to implement it would be plagiarism (because this is a homework
assignment.)
        # After I turn it in, I'll make this better.
(http://stackoverflow.com/questions/1740726/python-turn-string-into-operator)
        if (RpnArr[i] == "+"):
            ParseStk.append(B + A) # Order doesn't matter here, but putting B ahead of
A is consitent with the other conditions.
        elif (RpnArr[i] == "-"):
            ParseStk.append(B - A)
        elif (RpnArr[i] == "*"): # See immediately previous note.
            ParseStk.append(B * A)
        elif (RpnArr[i] == "/"):
            ParseStk.append(B // A) # Forcing integer division. Why? Because for this
project (Caesar Cypher) we will need integers so we can index the alphabet.
        elif (RpnArr[i] == "%"):
            ParseStk.append(B % A)
        elif (RpnArr[i] == "^"):
            ParseStk.append(B ** A)
    return ParseStk[0]

def CaesarCypher(message, infix):
    message = message.upper()
    CypherText = ""
    PostFix = InfixToPostfix(infix)
    for i in range(0, len(message)):
        if (message[i] in Alphabet):
            index = Alphabet.index(message[i])
            newIndex = ParseRPN(PostFix, index)
            CypherText += Alphabet[newIndex]
        else:
            CypherText += message[i] # If the character isn't part of our cypher, like a
punctuation mark or a space, we'll just ignore it.
    return CypherText

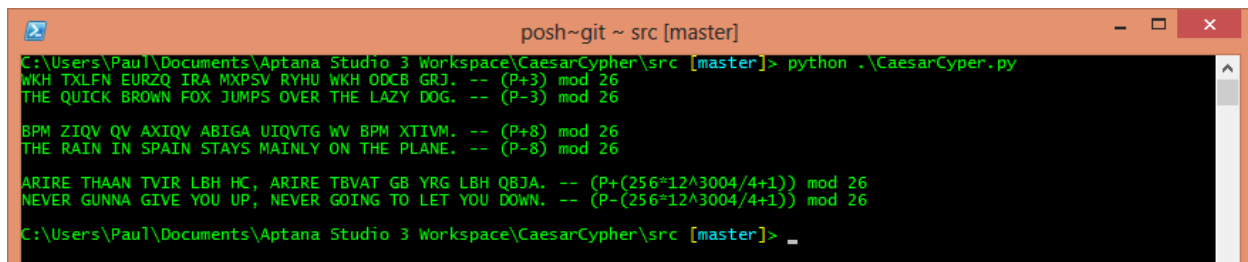
def DemoCC(message, key, oppKey):
    message = CaesarCypher(message, key)
    print("{} -- {}".format(message, key))
    message = CaesarCypher(message, oppKey)
    print("{} -- {}".format(message, oppKey))
    print()
    return

DemoCC("The quick brown fox jumps over the lazy dog.", "(P+3) mod 26", "(P-3) mod 26")

```

```
DemoCC("The rain in spain stays mainly on the plane.", "(P+8) mod 26", "(P-8) mod 26")
DemoCC("Never gunna give you up, never going to let you down.", "(P+(256*12^3004/4+1)) mod 26", "(P-(256*12^3004/4+1)) mod 26")
```

Screenshot of Output:



```
posh~git ~ src [master]
C:\Users\Paul\Documents\Aptana Studio 3 Workspace\CaesarCypher\src [master]> python .\CaesarCypher.py
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ. -- (P+3) mod 26
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG. -- (P-3) mod 26

BPM ZIQV QV AXIQV ABIGA UIQVGT WV BPM XTIVM. -- (P+8) mod 26
THE RAIN IN SPAIN STAYS MAINLY ON THE PLANE. -- (P-8) mod 26

ARIRE THAAN TVIR LBH HC, ARIRE TBVAT GB YRG LBH QBJA. -- (P+(256*12^3004/4+1)) mod 26
NEVER GUNNA GIVE YOU UP, NEVER GOING TO LET YOU DOWN. -- (P-(256*12^3004/4+1)) mod 26
C:\Users\Paul\Documents\Aptana Studio 3 Workspace\CaesarCypher\src [master]> _
```

WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ. -- (P+3) mod 26

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG. -- (P-3) mod 26

BPM ZIQV QV AXIQV ABIGA UIQVGT WV BPM XTIVM. -- (P+8) mod 26

THE RAIN IN SPAIN STAYS MAINLY ON THE PLANE. -- (P-8) mod 26

ARIRE THAAN TVIR LBH HC, ARIRE TBVAT GB YRG LBH QBJA. -- (P+(256*12^3004/4+1)) mod 26

NEVER GUNNA GIVE YOU UP, NEVER GOING TO LET YOU DOWN. -- (P-(256*12^3004/4+1)) mod 26