F20CN – Computer Network Security: Coursework 2

Group Number: 2

Student 1: Timothy Hayes (H00370889) (th2015)

Student 2: Zack Brown (H00372987) (zb2007)

Table of Contents

Task 1: Alternative Method of Public-Key Encryption

Description:

This program took the approach of delegating key tasks to functions such as the generation of each part of the private key, the public key, encryption, decryption and non-functional tasks such as converting between binary and string or read/writing to a file.

The input is read in by a path presented by the user, the program then generates random keys, then encrypts the input with the easy key to produce a cipertext which is then decrypted to present the plaintext. Each key attribute, ciphertext & plaintext is saved.

Pseudocode:

```
def convertToBinary(m):
        for i in m: binary.append(bin(i))
return binary
def convertToString(b)
        for i in b: string = string + char(i)
        return string
def getW(q):
do: w=randomInt while: gcd(w, q) != 1
return w
def generateEasyKey(m):
        for x in m: E.append(sum(e) + randomInt(>=1))
        return e
def generateHardKey(e, q, w)
return w.e(i) mod q of every i in e
def encrypt(m, h)
return sum(h(i)m(i) of every i in h
def decrypt(c, e, q, w):
cPrime = cw^-1 mod q
From e(n) to e(1) there is i:
if cPrime >= i: m(i) = 1 and cPrime = cPrime-1
Else: m(i) = 0
Return m
message = input("Enter file path")
m = convertToBinary(readFile(message)),
e = generateEasyKey(m.size()),
q = nextPrime(2en), w = getW(q),
h = generateHardKey(e, q, w)
```

```
ciphertext = encrypt(m, h),
recoveredtext = decrypt(ciphertext, e, q ,w)
writeFile(e, q, w, h, ciphertext, recoveredtext)
```

Task 1: Testing

To test whether the encryption and decryption algorithms function as intended, the original plaintext and the decrypted ciphertext can be compared. If the original plaintext and the decrypted ciphertext are a match, this means the encryption and decryption implementations are successful:

Input plaintext:

To Timothy Hayes:
A wonderful serenity has taken possession of my entire soul, like these sweet mornings of spring which I enjoy with my whole heart. I am alone, and feel the charm of existence in this spot, which was created for the bliss of souls like mine. I am so happy, my dear friend, so absorbed in the exquisite sense of mere tranquil existence, that I neglect my talents. I should be incapable of drawing a single stroke at the present moment; and yet I feel that I never was a greater artist than now.

Output deciphered ciphertext:

To Timothy Hayes:
A wonderful serenity has taken possession of my entire soul, like these sweet mornings of spring which I enjoy with my whole heart. I am alone, and feel the charm of existence in this spot, which was created for the bliss of souls like mine. I am so happy, my dear friend, so absorbed in the exquisite sense of mere tranquil existence, that I neglect my talents. I should be incapable of drawing a single stroke at the present moment; and yet I feel that I never was a greater artist than now.

These two strings are identical, and so both the encryption and decryption algorthims work as intended.

Additionally, the first four elements of the easy key e have been generated as follows: [1508, 3379, 6323, 11984…].

To test if this satisfies $e_i > \sum_{j=1}^{i-1} e_j$ for $1 \leq i \leq n$., we check e(4) > Σ(e(1) + e(2) + e(3)). So, we compute: (1508 + 3379 + 6323) = 11,210. As e(4) (11,984) is greater than 11,210 this proves the generation of e satisfies this condition.

The value for q generated by the test is a valid prime number as can be found in the appendix. And when performing gcd(w, q), we find that this results in 1, satisfying this condition specified in the specification.

As for h, this has been stored as a type "mpz" as per the import of gmpy2 found in the appendix. As a result, any of these values are too large to be presented here, but it may be worth noting that len(e) = len(h).

Gmpy2's next_prime method was used to significantly improve performance for finding q. This is due to the previous methodology's computationally expensiveness due to needing to check every number's primality following 2e(n) until a prime was found.

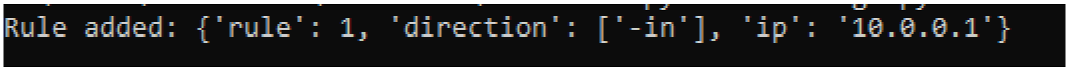
Task 2: Firewall Rules Application

Description & Pseudocode:

```
Function parse_arguments(command):
If no command or unrecognized command: Raise "Invalid Command"
For each expected argument in the command syntax:
if argument exists in the input:
Validate argument based on type:
if Valid, add to parameters
if Invalid, raise specific error
        Else if argument is mandatory: Raise "Missing Required Argument"
    If unused arguments remain: Raise "Unexpected argument"
    Assign default values for optional arguments
    Return parsed_parameters
    Function add_rule(new_rule):
    Format new_rule to ensure direction  always a list and to maintain structure
    Load rules
    If a rule with same IP exists:
    merge directions
    sort directions for consistent order
    Else:
    add new rule
    adjust priorities of existing rules
    sort ruleset by priority
    Save ruleset
    Function remove_rule(params):
    Load rules
    Find rule with specified priority
    If no matching rule found: print "no rule found"
            Return
        If a direction is provided:
            Remove the direction
            If no direction remain, remove the whole rule
        Else:
        Remove the whole rule
        Save the ruleset
    Function list_rules(params)
    Load rules
        For each rule in rules:
        If rule matches all specified parameters:
        add to filtered list
        Print filtered rules
        If no matches, print "No matching rules found"
```

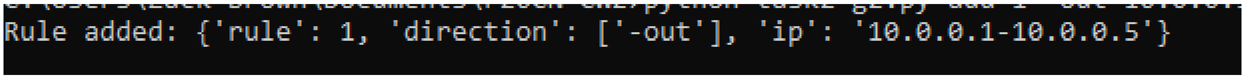
Task 2: Testing

add 1 -in 10.0.0.1



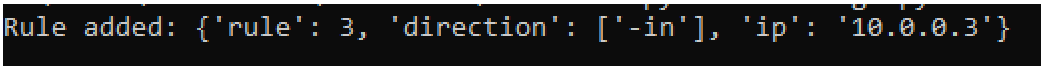
```
Rule added: {'rule': 1, 'direction': ['-in'], 'ip': '10.0.0.1'}
```

add 2 –out 10.0.0.1-10.0.0.5



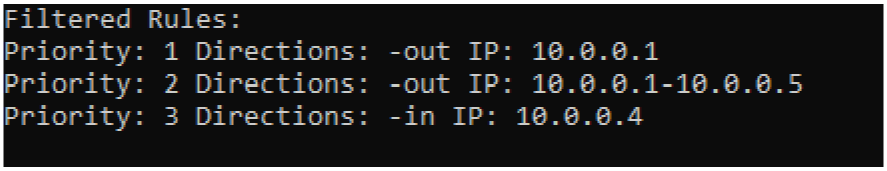
```
Rule added: {'rule': 1, 'direction': ['-out'], 'ip': '10.0.0.1-10.0.0.5'}
```

add 3 –in 10.0.0.4



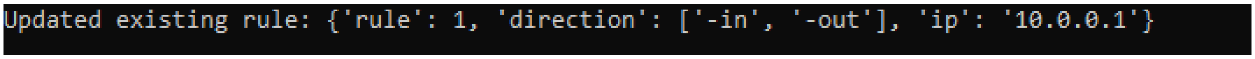
```
Rule added: {'rule': 3, 'direction': ['-in'], 'ip': '10.0.0.3'}
```

list



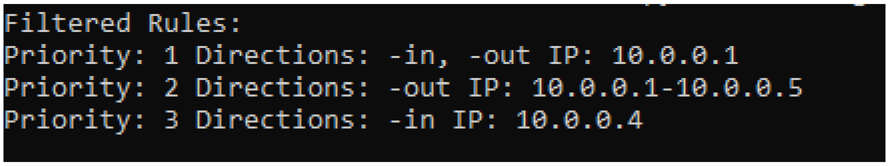
```
Filtered Rules:
Priority: 1 Directions: -out IP: 10.0.0.1
Priority: 2 Directions: -out IP: 10.0.0.1-10.0.0.5
Priority: 3 Directions: -in IP: 10.0.0.4
```

add 1 –out 10.0.0.1



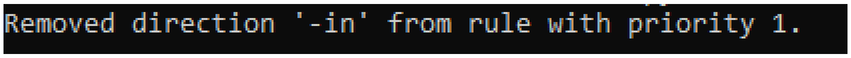
```
Updated existing rule: {'rule': 1, 'direction': ['-in', '-out'], 'ip': '10.0.0.1'}
```

list



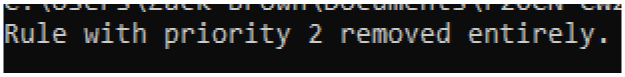
```
Filtered Rules:
Priority: 1 Directions: -in, -out IP: 10.0.0.1
Priority: 2 Directions: -out IP: 10.0.0.1-10.0.0.5
Priority: 3 Directions: -in IP: 10.0.0.4
```

remove 1 –in



```
Removed direction '-in' from rule with priority 1.
```

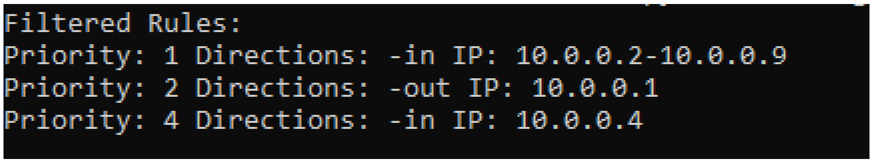
remove 2



```
Rule with priority 2 removed entirely.
```

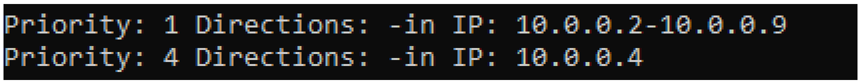
add –in 10.0.0.2-10.0.0.8

list



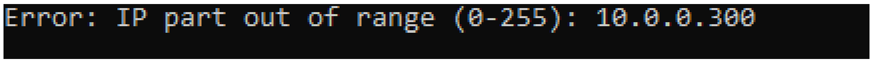
```
Filtered Rules:
Priority: 1 Directions: -in IP: 10.0.0.2-10.0.0.9
Priority: 2 Directions: -out IP: 10.0.0.1
Priority: 4 Directions: -in IP: 10.0.0.4
```

list –in



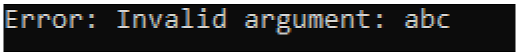
```
Priority: 1 Directions: -in IP: 10.0.0.2-10.0.0.9
Priority: 4 Directions: -in IP: 10.0.0.4
```

add 10.0.0.300



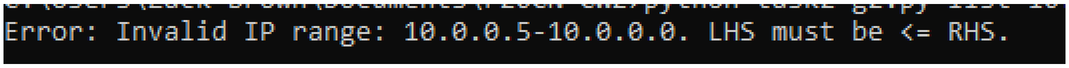
```
Error: IP part out of range (0-255): 10.0.0.300
```

add abc –in 10.0.0.1



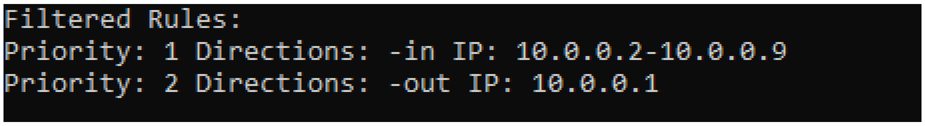
```
Error: Invalid argument: abc
```

list 10.0.0.5-10.0.0.0



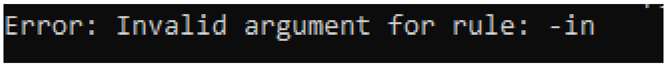
```
Error: Invalid IP range: 10.0.0.5-10.0.0.0. LHS must be <= RHS.
```

list 10.0.0.1-10.0.0.2



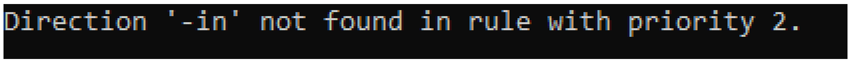
```
Filtered Rules:
Priority: 1 Directions: -in IP: 10.0.0.2-10.0.0.9
Priority: 2 Directions: -out IP: 10.0.0.1
```

remove –in



```
Error: Invalid argument for rule: -in
```

remove –in 2



```
Direction '-in' not found in rule with priority 2.
```

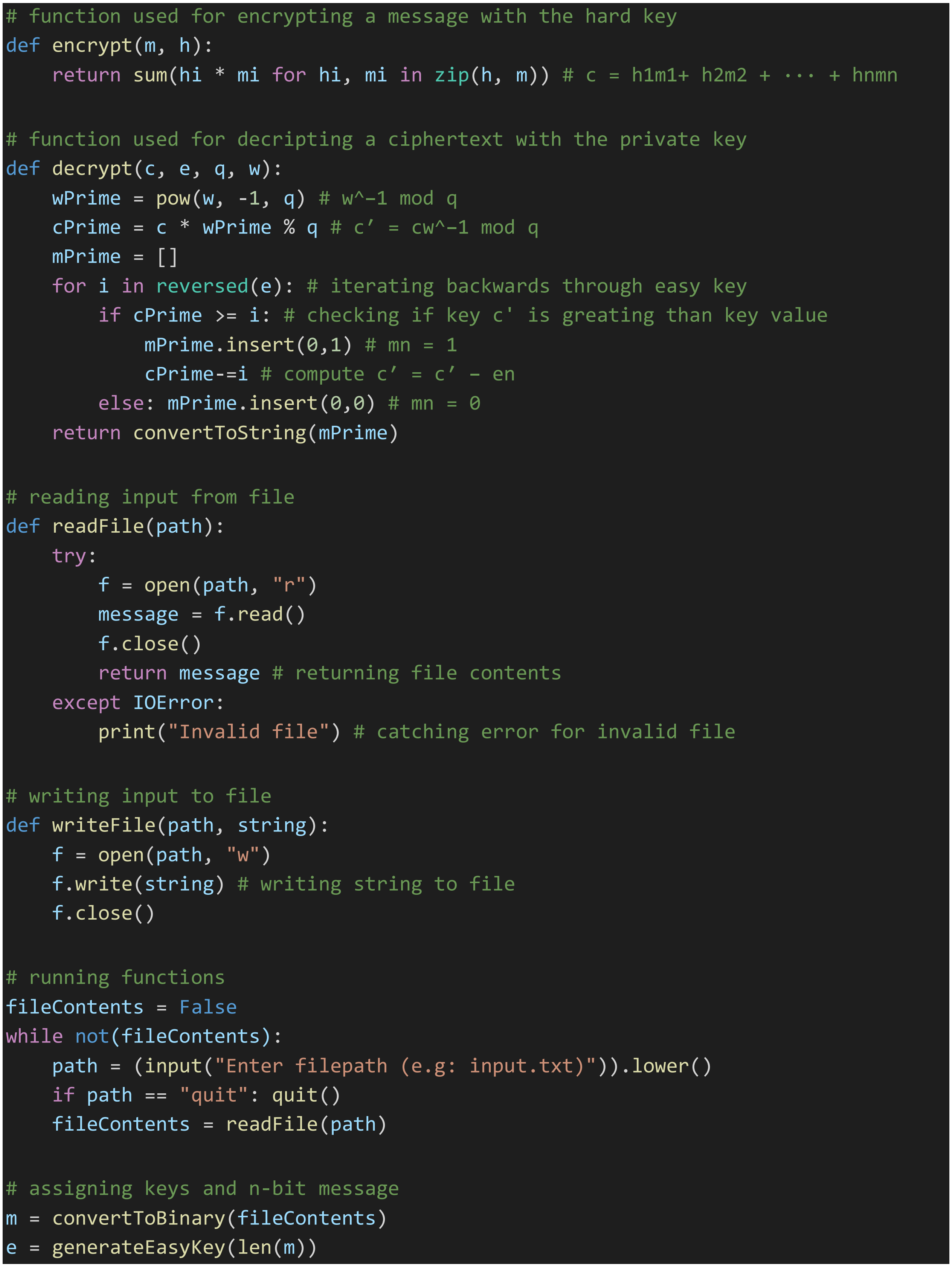
Summary

1. add:
    a. Successfully added rules with valid parameters
    b. Adjusts rule priorities as expected when a rule is added with the same priority as an existing rule
    c. Validates input and raises correct errors when an argument is not valid (ip and rule in the tests)
2. list:
    a. Displays all rules when no parameters are given
    b. Filters rules to display only rules with inputted parameters when any are given
    c. When an ip-range is given, lists rules that fall into or overlap with that range
    d. Validates input and raises correct errors when an argument is invalid (malformed ip range in the tests)
3. remove:
    a. Removes full rule when just rule is provided
    b. Successfully removes just direction from a rule when it is provided.
    c. When a direction is given and is the only direction for the rule, removes whole rule
    d. When a direction is given that is not in the rule, throws an error
    e. Validates input and raises correct errors (malformed rule parameter in the tests)

Appendix:

Task 1: Python Code

```
import math
import random
import gmpy2
# Case Van Horsen (2024). Welcome to gmpy2’s documentation! — gmpy2 2.2.1
documentation. [online] Available at: https://gmpy2.readthedocs.io/en/latest/.

# function for converting plaintext to n-bit message
def convertToBinary(m):
    binary = []  # Initialize list to hold all binary digits
    for char in m: binary.extend(int(digit) for digit in
bin(ord(char))[2:].zfill(8))
    return binary

# function for converting listing consisting of binary values into plaintext
def convertToString(b):
    return''.join(chr(int(("".join(str(x) for x in b))[i*8:i*8+8],2)) for i in
range(len(b)//8)) # every 8 binary values are converted into its string
counterpart to form the message

# function used for getting w in gcd(w, q) = 1
def getW(q):
    w = random.randint(0, 2*q) # generates random int between 0 & 2q
    if (math.gcd(w, q)) == 1: return w # base case: returns w checks if the gcd
is 1
    else: return getW(q) # recursive case: if gcd is not 1 then try a new w

# function for generating easy key (e)
def generateEasyKey(m):
    e = [] # initialising e as empty list
    for i in range(m): e.append(sum(e) + random.randint(1, int(m/2))) # creating
a key with same length as n-bit message and adding numbers which satisfy the
conditio
    return e

# function for generating hard key
def generateHardKey(e, q, w):
    return [(w * i) % q for i in e] # creating a key with same length as easy
key, following hi = w.ei mod q
```



```python
# function used for encrypting a message with the hard key
def encrypt(m, h):
    return sum(hi * mi for hi, mi in zip(h, m)) # c = h1m1+ h2m2 + ··· + hnmn

# function used for decripting a ciphertext with the private key
def decrypt(c, e, q, w):
    wPrime = pow(w, -1, q) # w^−1 mod q
    cPrime = c * wPrime % q # c’ = cw^−1 mod q
    mPrime = []
    for i in reversed(e): # iterating backwards through easy key
        if cPrime >= i: # checking if key c' is greating than key value
            mPrime.insert(0,1) # mn = 1
            cPrime-=i # compute c’ = c’ − en
        else: mPrime.insert(0,0) # mn = 0
    return convertToString(mPrime)

# reading input from file
def readFile(path):
    try:
        f = open(path, "r")
        message = f.read()
        f.close()
        return message # returning file contents
    except IOError:
        print("Invalid file") # catching error for invalid file

# writing input to file
def writeFile(path, string):
    f = open(path, "w")
    f.write(string) # writing string to file
    f.close()

# running functions
fileContents = False
while not(fileContents):
    path = (input("Enter filepath (e.g: input.txt)")).lower()
    if path == "quit": quit()
    fileContents = readFile(path)

# assigning keys and n-bit message
m = convertToBinary(fileContents)
e = generateEasyKey(len(m))
```

```
q = gmpy2.next_prime((2 * e[-1])+1) # used import for getting next prime: https://gmpy2.readthedocs.io/en/latest/ - Case Van Horsen (2024)
w = getW(q)
h = generateHardKey(e,q,w)
# encrypting and decrypting text
ciphertext = encrypt(m, h)
deciphertext = decrypt(ciphertext, e, q, w)

# writing keys, ciphertext, and decrypted text to file
writeFile("e.txt", str(e))
writeFile("q.txt", str(q))
writeFile("w.txt", str(w))
writeFile("h.txt", str(h))
writeFile("ciphertext.txt", str(ciphertext))
writeFile("deciphertext.txt", str(deciphertext))
```

Task 1: Screenshots

Plaintext:

To Timothy Hayes:

A wonderful serenity has taken possession of my entire soul, like these sweet mornings of spring which I enjoy with my whole heart. I am alone, and feel the charm of existence in this spot, which was created for the bliss of souls like mine. I am so happy, my dear friend, so absorbed in the exquisite sense of mere tranquil existence, that I neglect my talents. I should be incapable of drawing a single stroke at the present moment; and yet I feel that I never was a greater artist than now.

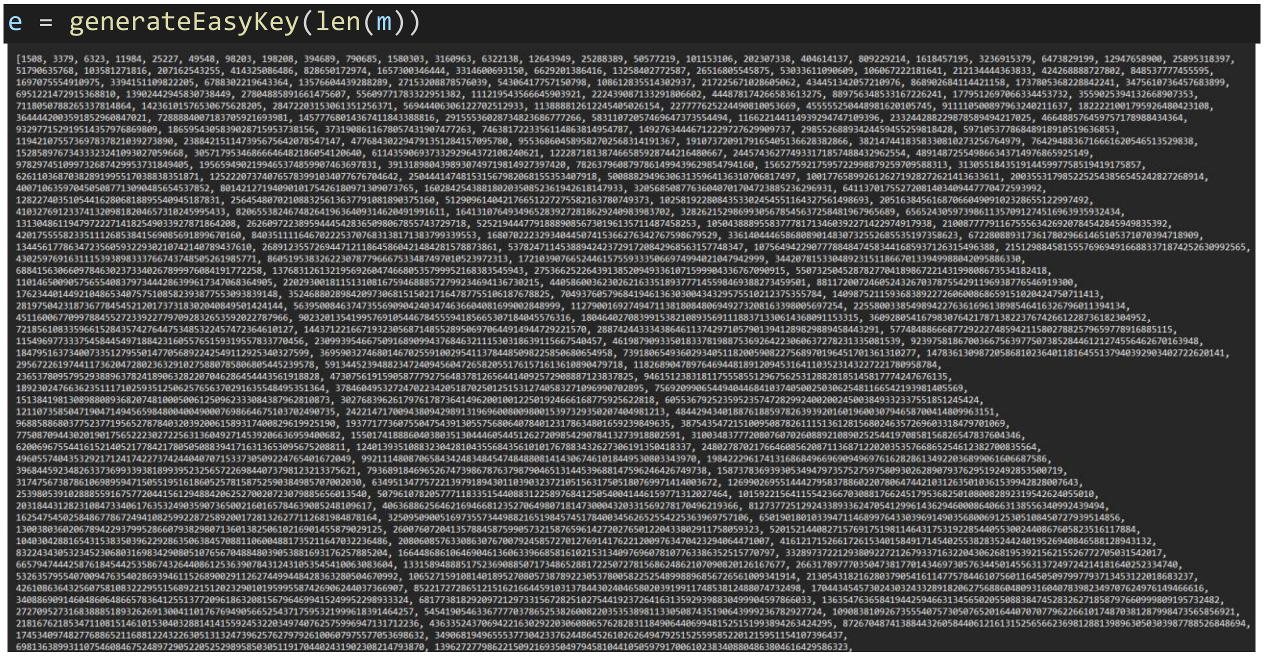
Ciphertext:

```
5748518501048419392338339510103850997038143166481505337197902774173527732
3300868826362168264169297180496736047460572816123077635235138270294573560
1192084601190160941691867357594197337413526511898671178305208152049629922
6066083728937205427174029935777969430532106353867688589045078479786377382
7965287340776473601256253976911152586609707197360114947442704864999975845
3107279707320067854197105504706875987778738377694658108752020133632066299
1057857619629512194190015270397596739477466548520594988779917494776958405
6007654219115222126038538323607097828131974769065239571023337548229510908
6143373442188793708592868118453687716582605719310770153320403004989181360
4109126678657086546163534564557782516492118028425254975448286382030950022
```

1160688595620139162480179649545048610320603544717998802960405924135225294
0038180575129833366234646116971170992409896590104043434723086167059731179
6574560626922391142684547601938587866712554739082091842530540971228207453
9940068334447946793324491457405200029608702311419091958357404749371475296
0237670917607007596087580237371544130513464577424763945531325070410219633
8806003934020222054111959229135213606922503985155455331770406036029201410
0642221992950825989435752370595976917905524207974354191168456091058O9

Key generation:

Private Key (e):



```
e = generateEasyKey(len(m))
```

```
[1508, 3379, 6323, 11984, 25227, 49548, 98203, 198208, 394689, 790685, 1580303, 3160963, 6322138, 12643949, 25288389, 50577219, 101153106, 202307338, 404614137, 809229214, 1618457195, 3236915379, 6473829199, 12947658900, 25895318397,
51790635768, 103581271816, 207162543255, 414325086486, 828650172974, 1657300346444, 3314600693150, 6629201386416, 13258402772587, 26516805545875, 53033611090609, 106067222181641, 212134444363833, 424268888727802, 848537777455595,
1697075554910975, 3394151109822205, 6788302219643364, 13576604439288289, 27153208878576039, 54306417757150798, 108612835514302937, 217225671028605062, 434451342057210976, 868902684114421158, 1737805368228842241, 3475610736457683899,
6951221472915368810, 13902442945830738449, 27804885891661475607, 55609771783322951382, 111219543566645903921, 222439087133291806602, 444878174266583613275, 889756348533167226241, 1779512697066334453732, 3559025394132668907353,
7118050788265337814864, 14236101576530675628205, 28472203153061351256371, 56944406306122702512933, 113888812612245405026154, 227777625224490810053669, 455555250448981620105745, 911110500897963240211637, 1822221001795926480423108,
3644442003591852960847021, 7288884007183705921693981, 14577768014367411843388816, 29155536028734823686777266, 58311072057469647373554494, 116622144114939294747109396, 233244288229878589494217025, 466488576459757178988434364,
932977152919514357976869809, 1865954305839028715953738156, 3731908611678057431907477263, 7463817223356114863814954787, 14927634446712229727629909737, 29855268893424459455259818428, 59710537786848918910519636853,
119421075573697837821039273890, 238842151147395675642078547147, 477684302294791351284157095780, 955368604589582702568314191367, 1910737209179165405136628382866, 3821474418358330810273256764979, 7642948836716661620546513529838,
15285897673433323241093027059668, 30571795346866646482186054120640, 61143590693733292964372108240621, 122287181387466585928744216480667, 244574362774933171857488432962554, 489148725549866343714976865925149,
978297451099732687429953731849405, 1956594902199465374859907463697831, 3913189804398930749719814927397420, 7826379608797861499439629854794160, 15652759217595722998879259709588313, 31305518435191445997758519419175857,
62611036870382891995517038838351871, 125222073740765783991034077676704642, 250444147481531567982068155353407918, 500888294963063135964136310706817497, 1001776589926126271928272621413633611, 2003553179852252543856545242827268914,
4007106359704505087713090485654537852, 8014212719409010175426180971309073765, 16028425438818020350852361942618147933, 32056850877636040701704723885236296931, 64113701755272081403409447770472593992,
128227403510544162806818895540945187831, 256454807021088325613637791081890375160, 512909614042176651227275582163780749373, 1025819228084353302454551164327561498693, 2051638456168706604909102328655122997492,
4103276912337413209818204657310245995433, 8206553824674826419636409314620491991611, 16413107649349652839272818629240983983702, 32826215298699305678545637258481967965689, 65652430597398611357091274516963935932434,
131304861194797222714182549033927871864208, 262609722389594445428365098067855743729718, 525219444779188890856730196135711487458253, 1050438889558377781713460392271422974917938, 2100877779116755563426920784542845949835392,
4201755558233511126853841569085691899670160, 8403511116467022253707683138171383799339553, 16807022232934044507415366276342767598679529, 33614044465868089014830732552685535197358623, 67228088931736178029661465105371070394718909,
134456177863472356059322930210742140789437610, 268912355726944712118645860421484281578873861, 537824711453889424237291720842968563157748347, 1075649422907778848474583441685937126315496388, 2151298845815557696949166883371874252630992565,
4302597691631115393898333766743748505261985771, 8605195383262230787796667533487497010523972313, 17210390766524461575593335066974994021047942999, 34420781533048923151186670133949988042095886330,
68841563066097846302373340267899976084191772258, 137683126132195692604746680535799952168383545943, 275366252264391385209493361071599904336767090915, 550732504528782770418986722143199808673534182418,
1101465009057565540837973444286399617347068364905, 2202930018115131081675946888572799234694136730215, 4405860036230262163351893777145598469388273459501, 8811720072460524326703787554291196938776546919300,
17623440144921048653407575108582393877553093839148, 35246880289842097306815150217164787755106187678825, 70493760579684194613630300434329575510212375355784, 140987521159368389227260600868659151020424750711413,
281975042318736778454521201737318302040849501424144, 563950084637473556909042403474636604081699002848999, 1127900169274947113818084806949273208163398005697254, 2255800338549894227636169613898546416326796011394134,
4511600677099788455272339227797092832653592022787966, 9023201354199576910544678455594185665307184045576316, 18046402708399153821089356911188371330614368091153315, 36092805416798307642178713822376742661228736182304952,
72185610833596615284357427644753485322457472364610127, 144371221667193230568714855289506970644914944729221570, 288742443334386461137429710579013941289829889458443291, 577484886668772922274859421158027882579659778916885115,
1154969773337545844549718842316055765159319557833770456, 2309939546675091689099437684632111530318639115667540457, 4619879093350183378198875369264223060637278231335081539, 9239758186700366756397750738528446121274556462670163948,
18479516373400733512795501477056892242549112925340327599, 36959032746801467025591002954113784485098225850680654958, 73918065493602934051182005908227568970196451701361310277, 147836130987205868102364011816455137940392903402722620141,
295672261974411736204728023632910275880785806805445239578, 591344523948823472409456047265820551761571613610890479718, 1182689047897646944818912094531641103523143227221780958784,
2365378095795293889637824189063282207046286454443561918828, 4730756191590587779275648378126564414092572908887123837825, 9461512383181175558551296756253128828185145817774247676135,
18923024766362351117102593512506257656370291635548495351364, 37846049532724702234205187025012515312740583271096990702895, 75692099065449404468410374050025030625481166542193981405569,
151384198130898808936820748100050061250962333084387962810873, 302768396261797617873641496200100122501924666168775925622818, 605536792523595235747282992400200245003849332337551851245424,
1211073585047190471494565984800400490007698664675103702490735, 2422147170094380942989131969600800980015397329350207404981213, 4844294340188761885978263939201601960030794658700414809963151,
9688588680377523771956527878403203920061589317400829619925190, 193771773607550475439130555756806407840123178634801659239849635, 387543547215100950878261115136128156802463572696033184797O1069,
77508709443020190175652223027225631360492714539206636959400682, 155017418886040380351304446054451262720985429078413273918802591, 310034837772080760702608892108902525441970858156826547837604346,
620069675544161521405217784217805050883941716313653095675208811, 1240139351088323042810435568435610101767883432627306191350418337, 2480278702176646085620871136871220203535766865254612382700835564,
4960557404353292171241742273742440407071533730509224765401672049, 9921114808706584342483484547484880814143067461018449530803343970, 19842229617413168684966969094969761628286134922036899061606687586,
39684459234826337369933938189939523256572269844073798123213375621, 79368918469652674739867876379879046513144539688147596246426749738, 158737836939305349479735752759758093026289079376295192492853500719,
317475673878610698959471505519516186052578158752590384985707002030, 634951347757221397918943011039032372105156317505180769971414003672, 1269902695514442795837886022078064744210312635010361539942828007643,
2539805391028885591675772044156129488420625270020723079885656013540, 5079610782057771183351544088312258976841250540041446159771312027464, 10159221564115542366703088176624517953682501080082892319542624055010,
20318443128231084733406176353249035907365002160165784639085248109617, 40636886256462169466812352706498071814730004320331569278170496219366, 81273772512924338933624705412996143629460008640663138556340992439494,
162547545025848677867249410825992287258920017281326277112681984878164, 325095090051697355734498821651984574517840034562652554225363969757106, 650190180103394711468997643303969149035680069125305108450727939514856,
1300380360206789422937995286607938298071360138250610216901455879029125, 2600760720413578845875990573215876596142720276501220433802911758059323, 5201521440827157691751981146431753192285440553002440867605823516117884,
10403042881654315383503962292863506384570881106004881735211647032236486, 20806085763308630767007924585727012769141762212009763470423294064471007, 41612171526617261534015849171454025538283524424019526940846588128943132,
83224343053234523068031698342908051076567048848039053881693176257885204, 166448686106469046136063396685816102153134097696078107763386352515770797, 332897372212938092272126793371632204306268195392156215526772705031542017,
665794744425876184544253586743264408612536390784312431053545410063083604, 1331589488851752369088507173486528817225072781568624862107090820126167677, 2663178977703504738177014346973057634450145563137249724214181640252334740,
5326357955407009476354028693946115268900291126274499448428363280504670992, 10652715910814018952708057387892230537800582252548999889685672656100934191 4, 21305431821628037905416114775784461075601164505097997793713453122018683237,
42610863643256075810832229551568922151202329010195995587426906244037366907, 85221727286512151621664459103137844302404658020391991174853812488074732498, 170443454573024303243328918206275688604809316040783982349707624976149466616,
340886909146048606486657836412551377209618632081567964699415249952298933324, 681773818292097212973315672825102754419237264163135929398830499904597866033, 1363547636584194425946631345650205508838474528326271858797660999809195732482,
2727095273168388851893262691300411017676949056652543717595321999618391464257, 5454190546336777703786525382600822035353898113305087435190643999236782927724, 10908381092673555407573050765201644070707796226610174870381287998473565856921,
21816762185347108151461015304032881414155924532203497407625759969471317122 36, 43633524370694221630292203060806576282831184906440699481525151993894263424295, 87267048741388443260584406121613152565662369812881398963050303987788526848694,
174534097482776886521168812243226305131324739625762797926100607975577053698632, 349068194965553773042337624486452610262649479251525595852201215951154107396437,
698136389931107546084675248972905220525298958503051191704402431902308214793870, 1396272779862215092169350497945810441050597917006102383408804863804616429586323,
```

Private Key (q):

```
q = gmpy2.next_prime((2 * e[-1])+1) # used import for getting next prime: https://gmpy2.readthedocs.io/en/latest/ - Case Van Horsen (2024)
```

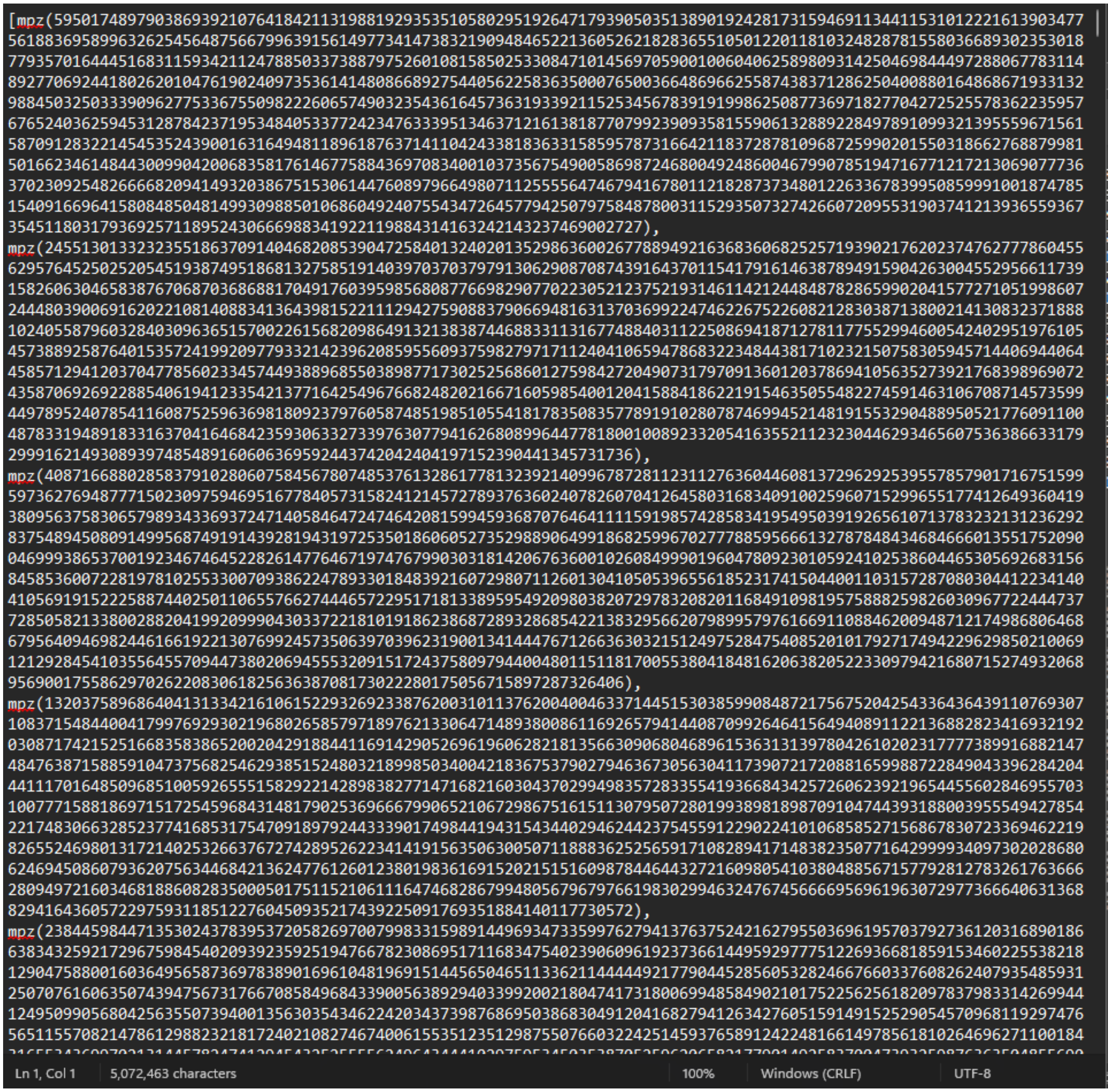
```
62968540265269800594565771216486184729621067957182104247654059122026867 29
```

```
6296854026526980059456577121648618472962106795718210424765405912202686729
8728061657618719780147378012217299341627534222185825253524768656988384899
3805965374146932990591007330650264686749239638899828139416239659674539151
1158078234238150347597609937202862373160949784438414469859417747371854577
3212939633035368549344544083333168615649468268354693878119787051587030404
6068438503789569052096127029918015862985468267782839001645073584096480795
6528931490596852327424328461470589490752606537785424107851494483896508109
8476819974297063022837057732446572920650962864609678123232763611659517412
6697447160490233194942120826808026750134706857067334464950818123369996002
5684373240027178535907743597075571425351893446792649498275058906077081006
8745974819115739513101722836545509536599965515534811543609282340734123682
7593914814639953465150771279022682791924807229297024327373242996690290428
2120906488883122298799409740689614564137161793200098261956998862562336549
6108028408176493623261132622441789478219823255176130641072119891742117272
5168280037935591973570071442691382185382466257626941546957242671169884161
4051222180768173203623215193715558515914456688852015834214182496940429501
9071085341688707857284487636653596735816081197805333578556078298 49
```

2037517168975884617689612840136385271638286079559515282436088102021689995
0976519957383205204185147037646548567307852375823398366477582O133

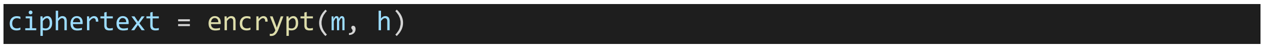
Public Key (h):

```
h = generateHardKey(e,q,w)
```

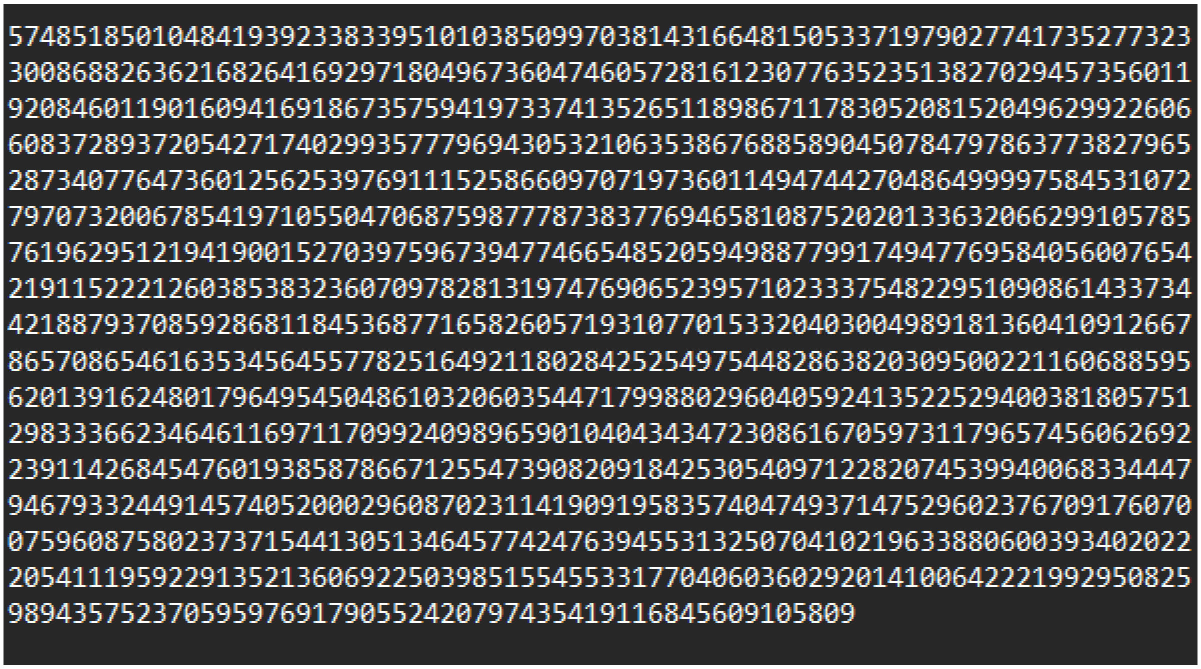


```
[mpz(5950174897903869392107641842113198819293535105802951926471793905035138901924281731594691134411531012221613903477
561883695899632625456487566799639156149773414738321909484652213605262182836551050122011810324828781558036689302353018
779357016444516831159342112478850337388797526010815850253308471014569705900100604062589809314250469844497288067783114
892770692441802620104761902409735361414808668927544056225836350007650036648696625587438371286250400880164868671933132
988450325033390962775336755098222606574903235436164573631933921152534567839191998625087736971827704272525578362235957
676524036259453128784237195348405337724234763339513463712161381877079923909358155906132889228497891099321395559671561
587091283221454535243900163164948118961876371411042433818363315859578731664211837287810968725990201550318662768879981
501662346148443009904200683581761467758843697083400103735675490058698724680049248600467990785194716771217213069077736
370230925482666682094149320386751530614476089796649807112555564746794167801121828737348012263367839950859991001874785
154091669641580848504814993098850106860492407554347264577942507975848780031152935073274266072095531903741213936559367
354511803179369257118952430666988341922119884314163242143237469002727),
mpz(24551301332323551863709140468208539047258401324020135298636002677889492163683606825257193902176202374762777860455
629576452502520545193874951868132758519140397037037979130629087087439164370115417916146387894915904263004552956611739
158260630465838767068703686881704917603959856808776698290770223052123752193146114212448487828659902041577271051998607
244480390069162022108140883413643981522111294275908837906694816313703699224746226752260821283038713800214130832371888
102405587960328403096365157002261568209864913213838744688331131677488403112250869418712781177552994600542402951976105
457388925876401535724199209779332142396208595560937598279717112404106594786832234844381710232150758305945714406944064
458571294120370477856023345744938896855038987717302525686012759842720490731797091360120378694105635273921768398969072
435870692692288540619412335421377164254967668248202166716059854001204158841862219154635055482274591463106708714573599
449789524078541160875259636981809237976058748519851055418178350835778919102807874699452148191553290488950521776091100
487833194891833163704164684235930633273397630779416268089964477818001008923320541635521123230446293465607536386633179
29991621493089397485489160606369592443742042404197152390441345731736),
mpz(40871668802858379102806075845678074853761328617781323921409967872811231127636044608137296292539557857901716751599
597362769487771502309759469516778405731582412145727893763602407826070412645803168340910025960715299655177412649360419
380956375830657989343369372471405846472474642081599459368707646411115919857428583419549503919265610713783232131236292
837548945080914995687491914392819431972535018606052735298890649918682599670277788595666132787848434684666013551752090
046999386537001923467464522826147764671974767990303181420676360010260849990196047809230105924102538604465305692683156
845853600722819781025533007093862247893301848392160729807112601304105053965561852317415044001103157287080304412234140
410569191522258874402501106557662744465722951718133895954920980382072978320820116849109819575888259826030967722444737
728505821338002882041992099904303372218101918623868728932868542213832956620798995797616691108846200948712174986806468
679564094698244616619221307699245735063970396231900134144476712663630321512497528475408520101792717494229629850210069
121292845410355645570944738020694555320915172437580979440048011511817005538041848162063820522330979421680715274932068
9569001755862970262208306182563638708173022280175056715897287326406),
mpz(13203758968640413133421610615229326923387620031011376200400463371445153038599084872175675204254336436439110769307
108371548440041799769293021968026585797189762133064714893800861169265794144087099264641564940891122136882823416932192
030871742152516683583865200204291884411691429052696196062821813566309068046896153631313978042610202317777389916882147
484763871588591047375682546293851524803218998503400421836753790279463673056304117390721720881659988722849043396284204
441117016485096851005926555158292214289838277147168216030437029949835728335541936684342572606239219654455602846955703
100777158818697151725459684314817902536966679906521067298675161511307950728019938981898709104744393188003955549427854
221748306632852377416853175470918979244333901749844194315434402946244237545591229022410106858527156867830723369462219
826552469801317214025326637672742895262234141915635063005071188836252565917108289417148382350771642999934097302028680
624694508607936207563446842136247761260123801983616915202151516098784464432721609805410380488567157792812783261763666
280949721603468188608283500050175115210611164746828679948056796797661983029946324767456666956961963072977366640631368
82941643605722975931185122760450935217439225091769351884140117730572),
mpz(23844598447135302437839537205826970079983315989144969347335997627941376375242162795503696195703792736120316890186
638343259217296759845402093923592519476678230869517116834754023906096192373661449592977751226936681859153460225538218
129047588001603649565873697838901696104819691514456504651133621144444921779044528560532824667660337608262407935485931
250707616063507439475673176670858496843390056389294033992002180474173180069948584902101752256256182097837983314269944
124950990568042563550739400135630354346224203437398768695038683049120416827941263427605159149152529054570968119297476
565115570821478612988232181724021082746740061553512351298755076603224251459376589124224816614978561810264696271100184
```

Encryption of Plaintext:

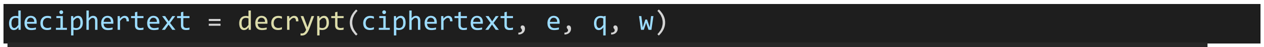


```
ciphertext = encrypt(m, h)
```

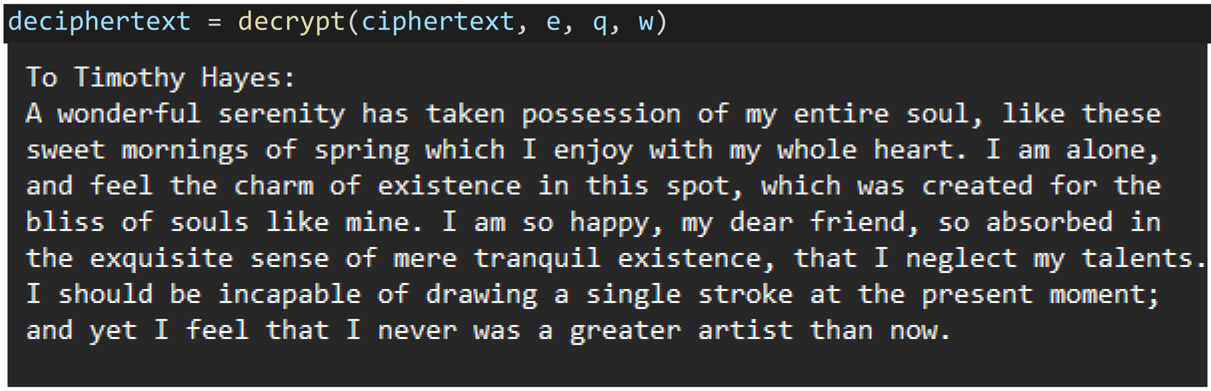


```
57485185010484193923383395101038509970381431664815053371979027741735277323
30086882636216826416929718049673604746057281612307763523513827029457356011
92084601190160941691867357594197337413526511898671178305208152049629922606
60837289372054271740299357779694305321063538676885890450784797863773827965
28734077647360125625397691115258660970719736011494744270486499997584531072
79707320067854197105504706875987778738377694658108752020133632066299105785
76196295121941900152703975967394774665485205949887799174947769584056007654
21911522212603853832360709782813197476906523957102333754822951090861433734
42188793708592868118453687716582605719310770153320403004989181360410912667
86570865461635345645577825164921180284252549754482863820309500221160688595
62013916248017964954504861032060354471799880296040592413522529400381805751
29833366234646116971170992409896590104043434723086167059731179657456062692
23911426845476019385878667125547390820918425305409712282074539940068334447
94679332449145740520002960870231141909195835740474937147529602376709176070
07596087580237371544130513464577424763945531325070410219633880600393402022
20541119592291352136069225039851554553317704060360292014100642221992950825
98943575237059597691790552420797435419116845609105809
```

Decryption of Ciphertext:



```
deciphertext = decrypt(ciphertext, e, q, w)
```

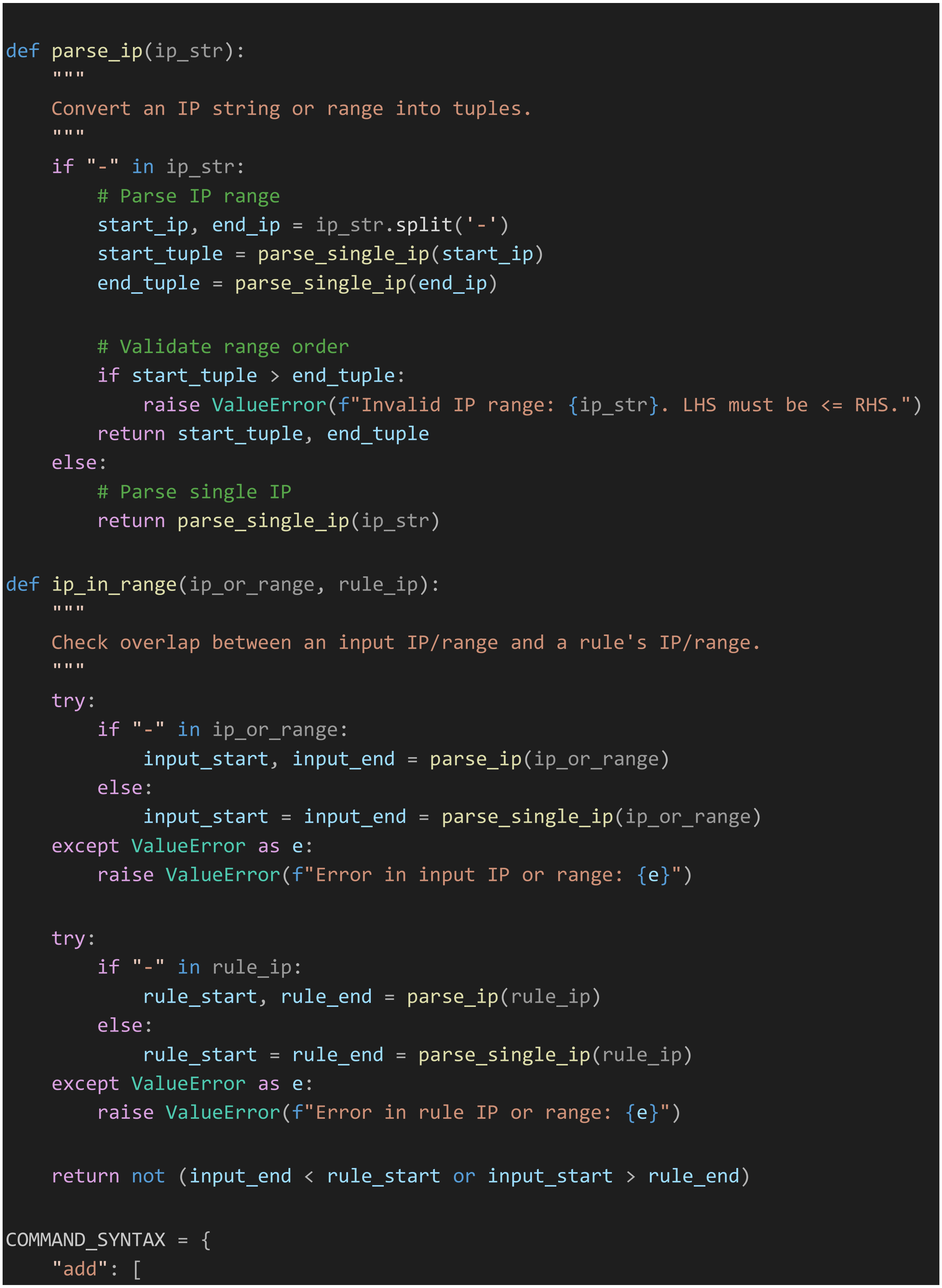


```
To Timothy Hayes:
A wonderful serenity has taken possession of my entire soul, like these
sweet mornings of spring which I enjoy with my whole heart. I am alone,
and feel the charm of existence in this spot, which was created for the
bliss of souls like mine. I am so happy, my dear friend, so absorbed in
the exquisite sense of mere tranquil existence, that I neglect my talents.
I should be incapable of drawing a single stroke at the present moment;
and yet I feel that I never was a greater artist than now.
```

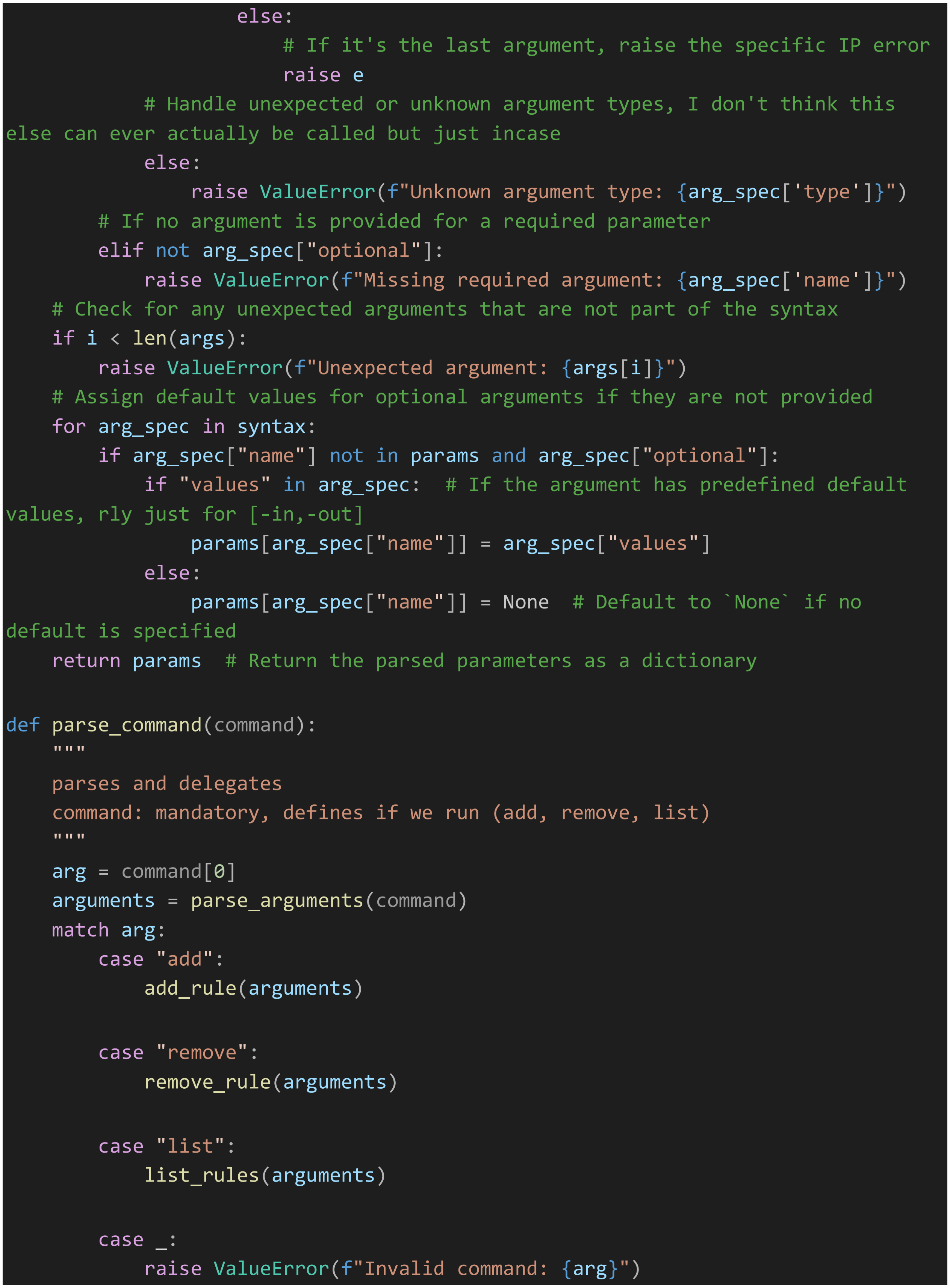
TASK 2 PYTHON CODE ON THE FOLLOWING PAGE!!!

Task 2: Python Code:

```python
import json # need some method to store rules between runs of the program, using a json file for this
import os # just for file operations
import sys # for command line input

RULESET_FILE = "ruleset.json"

def load_ruleset():
    """Load the ruleset from the file. Create an empty file if it doesn't exist."""
    if not os.path.exists(RULESET_FILE):
        # file not here, create it
        save_ruleset([])  # Save an empty list if file does not exist
    with open(RULESET_FILE, "r") as f:
        return json.load(f)

def save_ruleset(ruleset):
    """Save the ruleset to the file."""
    with open(RULESET_FILE, "w") as f:
        json.dump(ruleset, f, indent=4)

def validate_ip_part(part, ip_str):
    """Validate a single part of an IP address."""
    if not part.isdigit():
        raise ValueError(f"Invalid numeric part in IP address: {ip_str}")

    num = int(part)
    if num < 0 or num > 255:
        raise ValueError(f"IP part out of range (0-255): {ip_str}")

    return num

def parse_single_ip(ip_str):
    """Parse and validate a single IP address into a tuple."""
    parts = ip_str.split('.')
    if len(parts) != 4:
        raise ValueError(f"Invalid IP address format: {ip_str}")
    return tuple(validate_ip_part(part, ip_str) for part in parts)
```

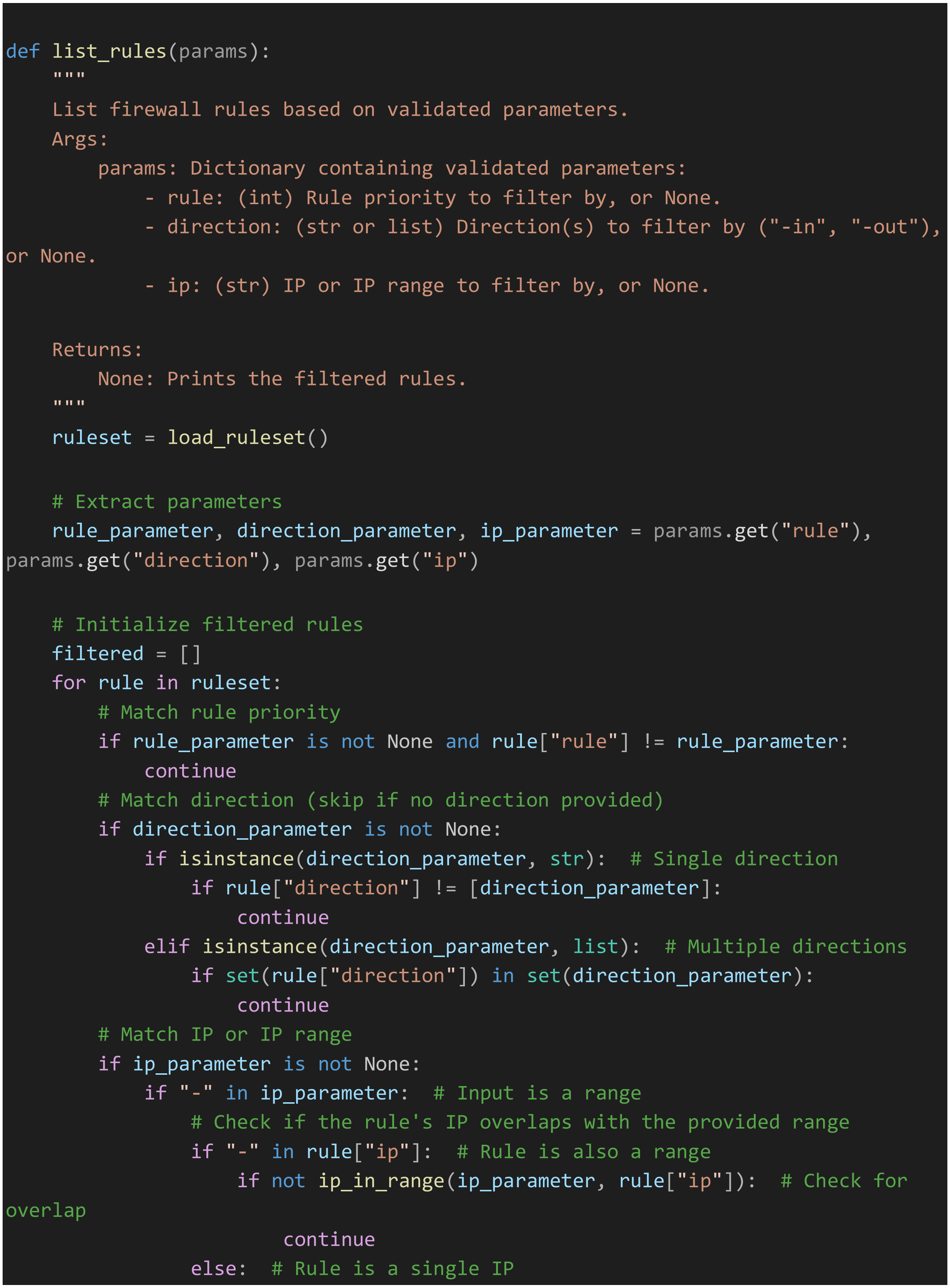


```
def parse_ip(ip_str):
    """
    Convert an IP string or range into tuples.
    """
    if "-" in ip_str:
        # Parse IP range
        start_ip, end_ip = ip_str.split('-')
        start_tuple = parse_single_ip(start_ip)
        end_tuple = parse_single_ip(end_ip)

        # Validate range order
        if start_tuple > end_tuple:
            raise ValueError(f"Invalid IP range: {ip_str}. LHS must be <= RHS.")
        return start_tuple, end_tuple
    else:
        # Parse single IP
        return parse_single_ip(ip_str)

def ip_in_range(ip_or_range, rule_ip):
    """
    Check overlap between an input IP/range and a rule's IP/range.
    """
    try:
        if "-" in ip_or_range:
            input_start, input_end = parse_ip(ip_or_range)
        else:
            input_start = input_end = parse_single_ip(ip_or_range)
    except ValueError as e:
        raise ValueError(f"Error in input IP or range: {e}")

    try:
        if "-" in rule_ip:
            rule_start, rule_end = parse_ip(rule_ip)
        else:
            rule_start = rule_end = parse_single_ip(rule_ip)
    except ValueError as e:
        raise ValueError(f"Error in rule IP or range: {e}")

    return not (input_end < rule_start or input_start > rule_end)

COMMAND_SYNTAX = {
    "add": [
        {"name": "rule", "type": "int", "values": 1,"optional": True},
        {"name": "direction", "type": "choice", "values": ["-in", "-out"], "optional": True},
        {"name": "ip", "type": "ip", "optional": False},
    ],
    "remove": [
        {"name": "rule", "type": "int", "optional": False},
        {"name": "direction", "type": "choice", "values": ["-in", "-out"], "optional": True},
    ],
    "list": [
        {"name": "rule", "type": "int", "optional": True},
        {"name": "direction", "type": "choice", "values": ["-in", "-out"], "optional": True},
        {"name": "ip", "type": "ip", "optional": True},
    ],
}

def parse_arguments(command):
    """
    General command parser based on defined syntax.
    Parses CLI arguments into a dictionary of parameters.

    Args:
        command: List of command-line arguments (e.g., ["add", "3", "-in", "10.0.0.1"]).

    Returns:
        dict: Parsed parameters.

    Raises:
        ValueError: If arguments do not match the expected syntax or are missing required arguments.
    """
    if not command:
        # Raise an error if no command is provided
        raise ValueError("No command provided.")

    cmd = command[0]  # Extract the command (e.g., "add", "remove", "list")
    args = command[1:]  # Extract the arguments following the command

    if cmd not in COMMAND_SYNTAX:
        # Raise an error if the command is not recognized
        raise ValueError(f"Invalid command: {cmd}")

    syntax = COMMAND_SYNTAX[cmd]  # Get the expected syntax for the command
    params = {}  # Dictionary to store parsed parameters
    i = 0  # Current argument index in the `args` list
    for arg_spec in syntax:
        # Iterate over the expected arguments as defined in the syntax
        if i < len(args):  # If there are remaining arguments to parse
            arg = args[i]  # Get the current argument
            # Handle `int` type arguments, for rule
            if arg_spec["type"] == "int":
                if arg.isdigit():  # Check if the argument is a valid integer
                    if int(arg) > 0: # rules cannot be less than 1
                        params[arg_spec["name"]] = int(arg)  # Add to the parsed parameters
                        i += 1  # Move to the next argument
                elif not arg_spec["optional"]:  # If the argument is required and invalid
                    raise ValueError(f"Invalid argument for {arg_spec['name']}: {arg}")
            # Handle `choice` type arguments e.g. direction parameter
            elif arg_spec["type"] == "choice":
                if arg in arg_spec["values"]:  # Check if the argument matches a valid choice
                    params[arg_spec["name"]] = arg  # Add to the parsed parameters
                    i += 1  # Move to the next argument
                elif not arg_spec["optional"]:  # If the argument is required and invalid, this never triggers for this arg type - purely here for good practice as it may be made required for another command
                    raise ValueError(f"Invalid {arg_spec['name']}: {arg}")
            # Handle `ip` type arguments (e.g., "10.0.0.1" or "10.0.0.1-10.0.0.5")
            elif arg_spec["type"] == "ip":
                try:
                    parse_ip(arg)  # Validate the IP address or range
                    params[arg_spec["name"]] = arg  # Add to the parsed parameters
                    i += 1  # Move to the next argument
                except ValueError as e:
                    # If there are more arguments, treat this as an invalid argument
                    if i + 1 < len(args):
                        raise ValueError(f"Invalid argument: {arg}")
```



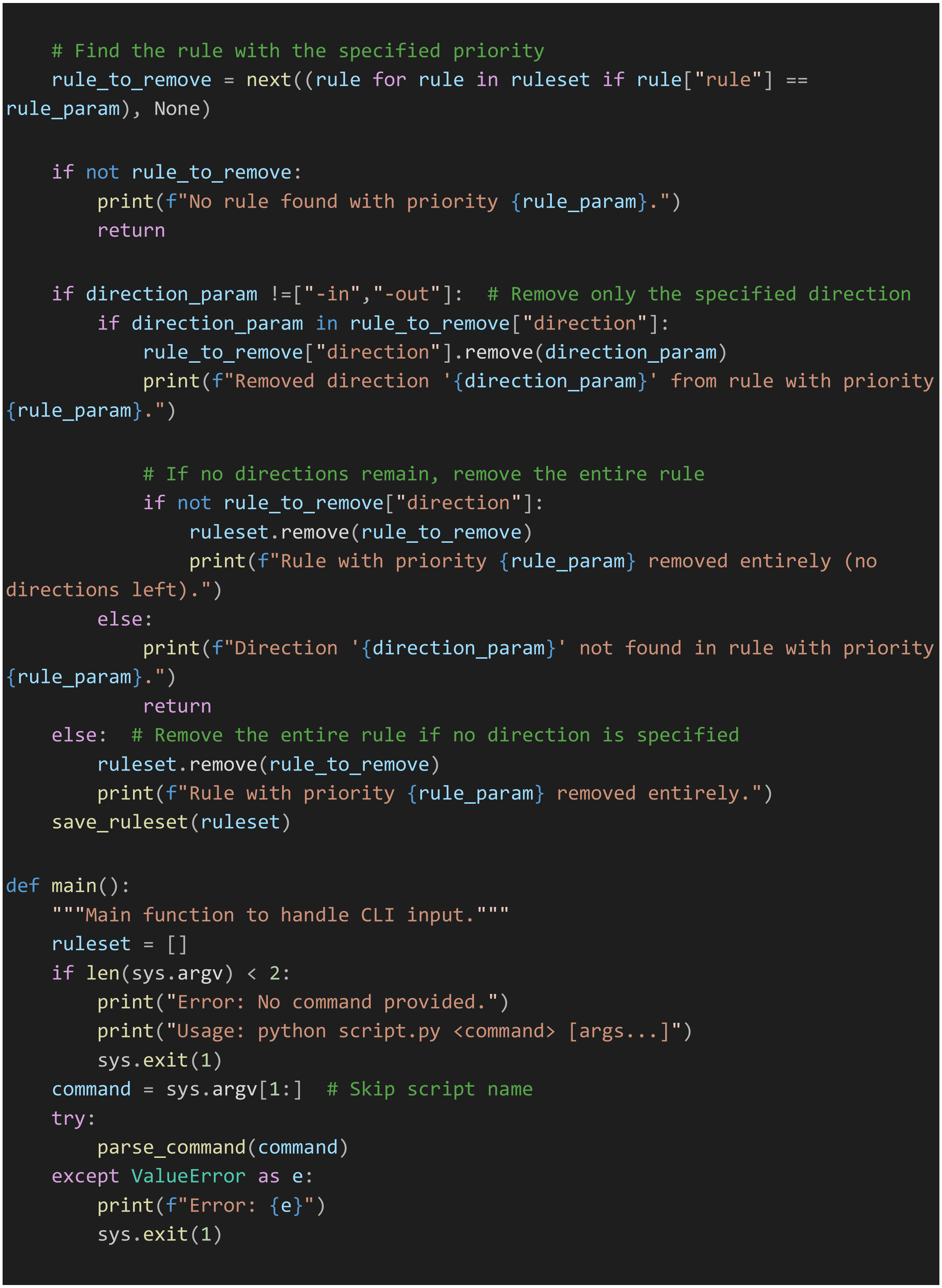
```
                    else:
                        # If it's the last argument, raise the specific IP error
                        raise e
            # Handle unexpected or unknown argument types, I don't think this
else can ever actually be called but just incase
            else:
                raise ValueError(f"Unknown argument type: {arg_spec['type']}")
        # If no argument is provided for a required parameter
        elif not arg_spec["optional"]:
            raise ValueError(f"Missing required argument: {arg_spec['name']}")
    # Check for any unexpected arguments that are not part of the syntax
    if i < len(args):
        raise ValueError(f"Unexpected argument: {args[i]}")
    # Assign default values for optional arguments if they are not provided
    for arg_spec in syntax:
        if arg_spec["name"] not in params and arg_spec["optional"]:
            if "values" in arg_spec:  # If the argument has predefined default
values, rly just for [-in,-out]
                params[arg_spec["name"]] = arg_spec["values"]
            else:
                params[arg_spec["name"]] = None  # Default to `None` if no
default is specified
    return params  # Return the parsed parameters as a dictionary

def parse_command(command):
    """
    parses and delegates
    command: mandatory, defines if we run (add, remove, list)
    """
    arg = command[0]
    arguments = parse_arguments(command)
    match arg:
        case "add":
            add_rule(arguments)

        case "remove":
            remove_rule(arguments)

        case "list":
            list_rules(arguments)

        case _:
            raise ValueError(f"Invalid command: {arg}")
```

```python
def add_rule(new_rule):
    """
    rule(priority): int, if none - rule = 1 by default,
    [-in|-out]: optional, defaults to -in and -out,
    addr: mandatory, raise ValueError if not supplied or incorrect.

    """
    new_rule = { # maintains the order of rules (without this when rule defaults to 1 it gets put at the back of the rule)
        'rule': new_rule['rule'],
        'direction': [new_rule['direction']] if isinstance(new_rule['direction'], str) else new_rule['direction'],
        'ip': new_rule['ip']
    }

    print(new_rule['rule'])
    ruleset = load_ruleset()

    # Check if a rule with the same IP already exists
    existing_rule = next((rule for rule in ruleset if rule["ip"] == new_rule["ip"]), None)

    if existing_rule:
        # Merge directions if IP exists
        for dir_ in new_rule["direction"]:
            if dir_ not in existing_rule["direction"]:
                existing_rule["direction"].append(dir_)
                existing_rule["direction"].sort(key=lambda x: ["-in", "-out"].index(x)) # sorts so in appears first (could hardcode to become -in -out if we update rule, but bad practice)
                print(f"Updated existing rule: {existing_rule}")
                save_ruleset(ruleset)
            return
    ruleset.append(new_rule)
    # Adjust priorities to fit the new rule
    for rule in ruleset:
        if rule["rule"] >= new_rule["rule"] and rule != new_rule:
            rule["rule"] += 1

    ruleset.sort(key=lambda rule: rule["rule"])  # Sort ruleset by priority
    save_ruleset(ruleset)
    print(f"Rule added: {new_rule}")
```



```python
def list_rules(params):
    """
    List firewall rules based on validated parameters.
    Args:
        params: Dictionary containing validated parameters:
            - rule: (int) Rule priority to filter by, or None.
            - direction: (str or list) Direction(s) to filter by ("-in", "-out"), or None.
            - ip: (str) IP or IP range to filter by, or None.

    Returns:
        None: Prints the filtered rules.
    """
    ruleset = load_ruleset()

    # Extract parameters
    rule_parameter, direction_parameter, ip_parameter = params.get("rule"), params.get("direction"), params.get("ip")

    # Initialize filtered rules
    filtered = []
    for rule in ruleset:
        # Match rule priority
        if rule_parameter is not None and rule["rule"] != rule_parameter:
            continue
        # Match direction (skip if no direction provided)
        if direction_parameter is not None:
            if isinstance(direction_parameter, str):  # Single direction
                if rule["direction"] != [direction_parameter]:
                    continue
            elif isinstance(direction_parameter, list):  # Multiple directions
                if set(rule["direction"]) in set(direction_parameter):
                    continue
        # Match IP or IP range
        if ip_parameter is not None:
            if "-" in ip_parameter:  # Input is a range
                # Check if the rule's IP overlaps with the provided range
                if "-" in rule["ip"]:  # Rule is also a range
                    if not ip_in_range(ip_parameter, rule["ip"]):  # Check for overlap
                        continue
                else:  # Rule is a single IP
```

```
                    rule_ip_tuple = parse_ip(rule["ip"])
                    input_start, input_end = parse_ip(ip_parameter)
                    if not (input_start <= rule_ip_tuple <= input_end):  # Single
IP must fall in range
                        continue
            else:  # Input is a single IP
                # Check if the single IP matches the rule
                if "-" in rule["ip"]:  # Rule is a range
                    if not ip_in_range(rule["ip"], ip_parameter):  # Check if
single IP falls in range
                        continue
                else:  # Rule is also a single IP
                    if rule["ip"] != ip_parameter:  # Direct match required
                        continue
        # If rule matches all conditions, add it to filtered list
        filtered.append(rule)

    # Print filtered results
    print("Filtered Rules:")
    if filtered:
        for rule in filtered:
            directions = ", ".join(rule["direction"])
            print(f"Priority: {rule['rule']} Directions: {directions} IP:
{rule['ip']}")
    else:
        print("No matching rules found.")

def remove_rule(params):
    """
    Remove a firewall rule based on validated parameters.

    Args:
        params: Dictionary containing validated parameters:
            - rule: (int) Priority of the rule to remove. Mandatory.
            - direction: (str) Direction to remove ("-in" or "-out"), or None.

    Returns:
        None: Modifies the ruleset in place.
    """
    ruleset = load_ruleset()

    rule_param = params.get("rule")
    direction_param = params.get("direction")
```



```
    # Find the rule with the specified priority
    rule_to_remove = next((rule for rule in ruleset if rule["rule"] ==
rule_param), None)

    if not rule_to_remove:
        print(f"No rule found with priority {rule_param}.")
        return

    if direction_param !=["-in","-out"]:  # Remove only the specified direction
        if direction_param in rule_to_remove["direction"]:
            rule_to_remove["direction"].remove(direction_param)
            print(f"Removed direction '{direction_param}' from rule with priority
{rule_param}.")

            # If no directions remain, remove the entire rule
            if not rule_to_remove["direction"]:
                ruleset.remove(rule_to_remove)
                print(f"Rule with priority {rule_param} removed entirely (no
directions left).")
        else:
            print(f"Direction '{direction_param}' not found in rule with priority
{rule_param}.")
            return
    else:  # Remove the entire rule if no direction is specified
        ruleset.remove(rule_to_remove)
        print(f"Rule with priority {rule_param} removed entirely.")
    save_ruleset(ruleset)

def main():
    """Main function to handle CLI input."""
    ruleset = []
    if len(sys.argv) < 2:
        print("Error: No command provided.")
        print("Usage: python script.py <command> [args...]")
        sys.exit(1)
    command = sys.argv[1:]  # Skip script name
    try:
        parse_command(command)
    except ValueError as e:
        print(f"Error: {e}")
        sys.exit(1)
```

```
if __name__ == "__main__":
    main()
```