DECRYPT MASTER

Major project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Nikhil Naresh (181430)
Miguel Kundal (181204)

UNDER THE SUPERVISON OF
Dr. Kapil Sharma

(Department of Computer Science & Engineering and Information Technology)

Jaypee University of Information Technology,

Waknaghat, 173234, Himachal Pradesh, INDIA

□

TABLE OF CONTENT

CONTENT PAGE NO.

DECLARATION

We hereby declare that; this project has been done by our group under the supervision of Dr. Kapil Sharma, Jaypee University of Information Technology. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:
Dr. Kapil Sharma
Assistant Professor
(Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology)

Submitted by:
Nikhil Naresh (181430)
Miguel Kundal (181204)
(Computer Science & Engineering Department
Jaypee University of Information Technology)

CERTIFICATE

This is to certify that the work which is being presented in the project report titled “**Decrypt Master”** in partial fulfilment of the requirements for the award of the degree of B. Tech in Computer Science and Engineering and submitted to the Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by “**Nikhil Naresh (181430), Miguel Kundal (181204)”** during the period from July 2021- Dec 2021 & Jan 2022 - June 2022 under the supervision of **Dr. Kapil Sharma**, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Submitted by:
Nikhil Naresh (181430)
Miguel Kundal (181204)
Computer Science & Engineering Department
Jaypee University of Information Technology

(The above statement made is correct to the best of my knowledge.)

Supervised by:
Dr. Kapil Sharma
Assistant Professor (SG)
(Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology)

ACKNOWLEDGEMENT

Firstly, we express our heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the project work successfully.

We are really grateful and wish my profound my indebtedness to Supervisor
Dr. Kapil Sharma,
Assistant Professor (SG),
(Department of CSE Jaypee University of Information Technology)

Deep Knowledge & keen interest of my supervisor in the field of "**Information Security"** to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. Kapil Sharma,** Department of CSE, for his kind help to finish this project.

We would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, we want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

Finally, we must acknowledge with due respect the constant support and patience of my parents.

Nikhil Naresh (181430)
Miguel Kundal (181204)
(Computer Science & Engineering Department
Jaypee University of Information Technology)

1. Introduction

1.1 Introduction

It has evolved into an integrated layer of protection for all digital transformation activities, which are now collectively known as digital business. As the core of modern security systems, cryptography is used to secure transactions and communications, safeguard personally identifiable information (PII) and other private data, verify identity, prevent document tampering, and develop server trust. Cryptography is one of the most important approaches used by organizations to protect the systems that store their most precious asset - data – whether it is in transit or at rest. Data includes personally identifiable information (PII), employee PII, intellectual property, corporate strategies, and any other private information. Since a result, cryptography is critical infrastructure, as sensitive data security increasingly relies on cryptographic solutions.

When sensitive data is wrapped in the invisible layers that make up cryptography, it becomes unreadable and unmodifiable, preventing bad actors from committing crimes. Algorithms, keys, libraries, and certificates are the basic parts that keep the cryptographic levels safe, as stated here:

To safeguard sensitive information, cryptographic keys are used in conjunction with cryptographic algorithms. To be effective, cryptographic keys must have an adequate key length as determined by the National Institute of Standards and Technology (NIST), and private keys must be kept secret. Cryptography becomes obsolete when insecure keys are utilized or secret keys are revealed.

Digital Certificates are used to preserve trust between connected digital components. To eliminate security breaches, digital certificates must be appropriately managed to ensure the usage of compliance algorithms and key lengths, as well as being renewed prior to expiration. Massive system disruptions or data breaches might result from non-compliant or concealed certificates.

1.2 Problem Statement

- In modern day cryptography we use different available encoding schemes to encrypt a text.
- A single encoding scheme encryption can be easily handled but problem arises when multiple encoding schemes are faced/ unknown encoding schemes are encountered.
- It will be very difficult for a cryptographer to deal with such encrypted text.
- A tool which can decode multiple encoding scheme and at the same time identify the base encoding scheme will help the cryptographer in decoding any text he wants!

1.3 Objectives

- Our main objective is to reduce time consumed by decoding the encoded text for the Cryptographer.
- In detail is to make a unique tool that can decode any alphanumeric encoding schemes.
- This tool will be also available as a library in python to use decryption and encryption any time.
- This tool will accept single user input, multiple use inputs, single encoded bases as well as multi-encoded base.
- This tool will also predict the encoding scheme of our encrypted text

1.4 Methodology

- This tool will be written in python with help of various libraries and packages.
- Encoding schemes of different bases are integrated together to work according the encrypted text.
- Encrypted text will be scanned for encoding schemes and decrypted text will be produced as soon as encoding scheme is identified.
- In case of multiple scheme this tool has a magic mode, which will decode the text with help of multiple decoding schemes.

2. Literature Review

- **Wojciech Muła, Daniel Lemire, "Base64 encoding and decoding at almost the speed of a memory copy".**

Base64 code consists of 64 ASCII characters, which include all 26 letters (upper and lower case), all ten digits, and two extra characters ('+' and '/'). Each of these 64 letters represents a 6-bit unsigned integer value between 0 and 255.
We show how, on modern Intel processors, we can encode and decode base64 data at nearly the same speed as a memory copy (memcpy), despite the fact that the data in the first-level (L1) cache does not fit.

- **Kenang Eko Prasetyo, Tito Waluyo Purboyo and Randy Erfa Saputra, "A Survey on Data Compression and Cryptographic Algorithms".**

Data security and confidentiality are significant concerns for every organization, whether it is a business, an institution, or a government agency, as well as for individuals. Especially if the info rmation is kept on a computer network that is linked to the internet or a public network. The ability of an organization to gather and communicate information in a timely and accurate manner will have a substantial influence. In this study, we will use a sort of encryption to define the message/data delivery security system, with the goal of ensuring data or message secrecy. So that people who aren't qualified can't see or read the information we send out. We'll employ one of them here, which is a technique of security system that uses the Cryptographic algorithm, because many security systems are used by organizations and individuals.

- **S. Josefsson, "The Base16, Base32, and Base64 Data Encodings"**

This document covers the base 64, base 32, and base 16 encoding algorithms. Line feeds, padding, and non-alphabet characters in encoded data, as well as encoding alphabets and canonical encodings, are all covered.

The Base 32 encoding is designed to represent arbitrary octet sequences in a case-insensitive but not always human-readable format. A 33-character subset of US-ASCII is used, allowing for the expression of 5 bits per printed character.

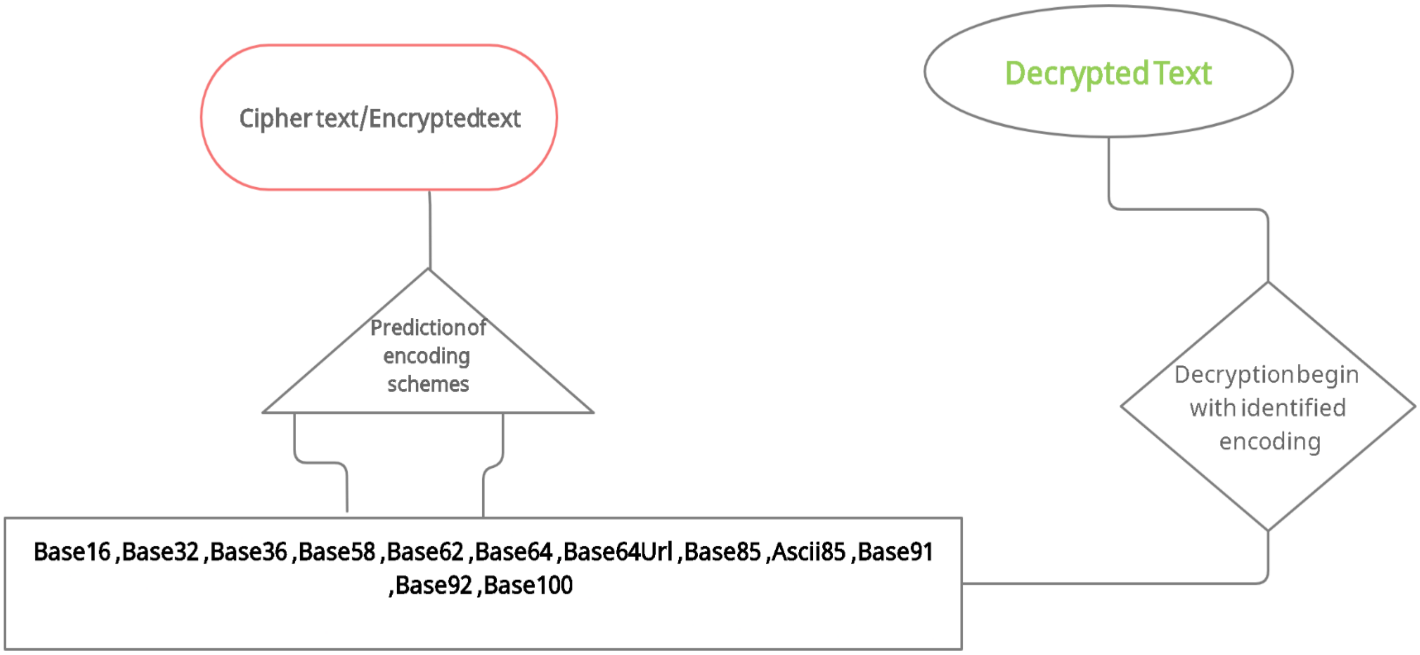
The typical case-insensitive hex encoding is Base 16 encoding, also known as "base16" or "hex." A 16-character subset of US-ASCII is used, allowing for the representation of four bits per printed character.

- **Mohammad A. Ahmad, Imad Fakhri Al Shaikhli, Hanady Mohammad Ahmad," Protection of the Texts Using Base64 and MD5".**

Encryption is a mathematical and computer-based process. Cryptography is a set of techniques and tactics for turning data into an unreadable and incomprehensible format for anyone who does not have the authority to read or write on it. The basic purpose of encrypting data and information is to protect it while protecting privacy. A base64 encryption strategy is presented in this work, which is a collection of encoding schemes that convert binary data into a sequence of ASCII codes. In addition, the Base64-encrypted file is hashed using the MD5 hash method.

3. System Development

a. <u>Design:</u>


Ciphertext/Encryptedtext
Predictionof encoding schemes
Base16 ,Base32 ,Base36 ,Base58 ,Base62 ,Base64 ,Base64Url ,Base85 ,Ascii85 ,Base91 ,Base92 ,Base100
Decryptionbegin with identified encoding
DecryptedText

(The following flow chart explains the design of our algorithm which explains working of the tool)

Step 1: First cipher text is scanned by the tool

Step 2: Encoding scheme is predicted from the cipher text

Step 3: In case of multiple encoding, magic mode is executed

Step 4: Decryption begins with identified encoding schemes

Step 5: Decryption ends

b. Model:

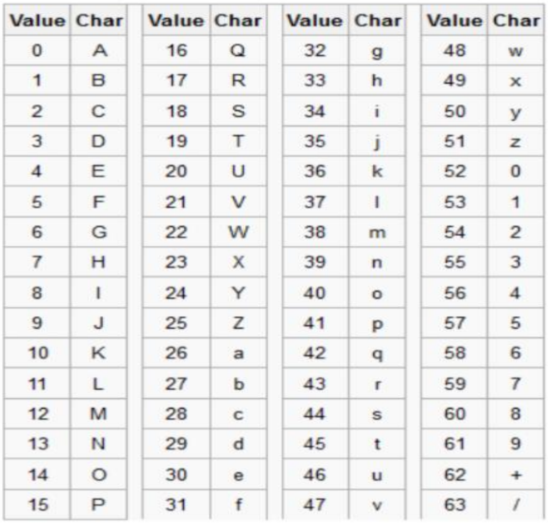
Decryption of an encoding scheme:

As an example, working methodology of base64 and base 36 encryption is explained:

Base 64

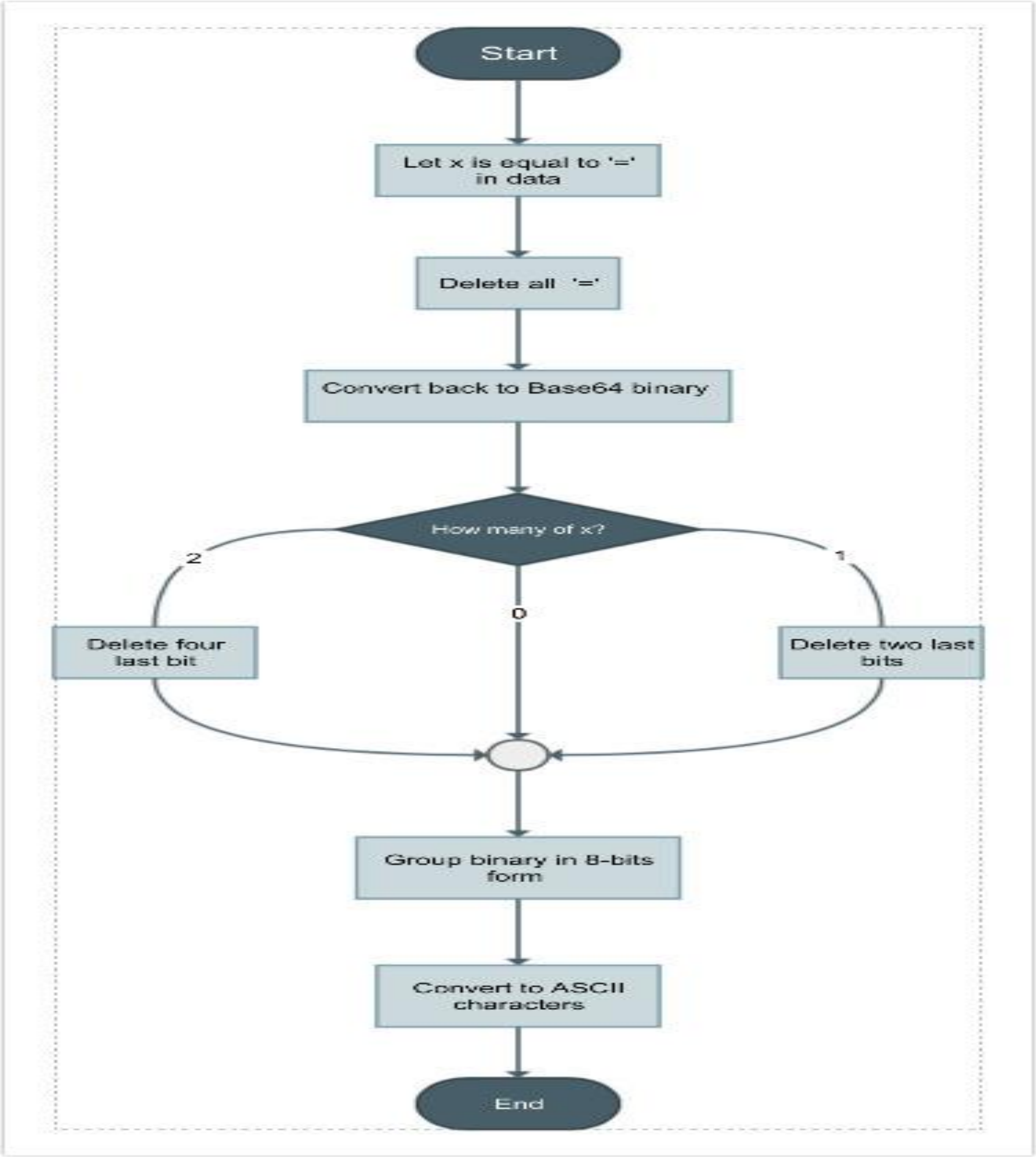
This scheme breaks the binary data into 6-bit segments of 3 bytes and represent those characters into ASCII standard. It does that in essentially two steps.

• The binary string must first be broken down into 6-bit units. To maintain the integrity of the sentence, Base64 is limited to using only 6 bits (which is 266 = 64 characters). The 64 characters (hence the name Base64) are 10 numerals, 26 lowercase characters, 26 capital characters, as well as the Plus sign (+) and the Forward Slash (/). The Equal sign (=) is the 65th character, which is known as a pad. When the last segment of binary data does not contain all six bits, this character is used.



Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

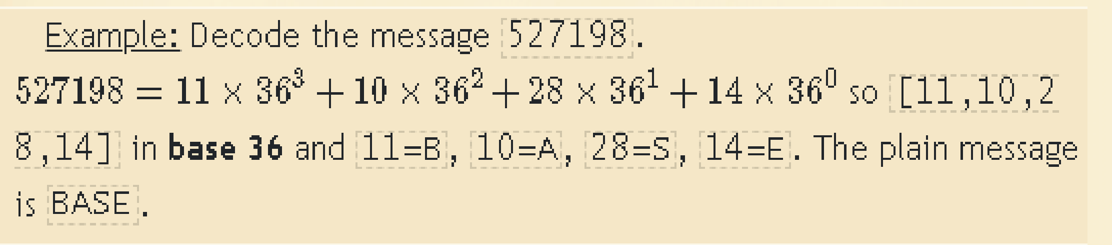
Fig. 1 Base64 characters table


Start
Let x is equal to '=' in data
Delete all '='
Convert back to Base64 binary
How many of x?
2
1
0
Delete four last bit
Delete two last bits
Group binary in 8-bits form
Convert to ASCII characters
End

(By a flowchart we can easily explain the base 64 encryption)

Base 36

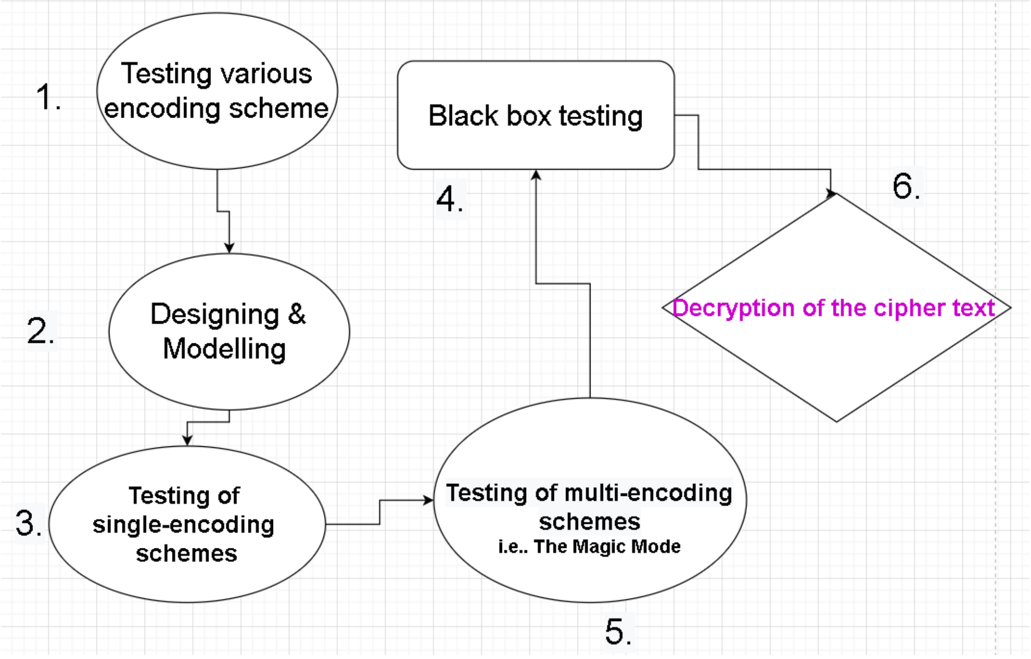
This encoding scheme base consists of 36 alphabetical characters including the 26 letters of the alphabet and the 10 digits. Any number can be converted to base 36, and any word can be converted to base 10.



Example: Decode the message `527198`.
$527198 = 11 \times 36^3 + 10 \times 36^2 + 28 \times 36^1 + 14 \times 36^0$ so `[11,10,28,14]` in **base 36** and `11=B`, `10=A`, `28=S`, `14=E`. The plain message is `BASE`.

c. Development:

The development of this project happened in 6 states as shown in below flowchart:


1.
Testing various encoding scheme
2.
Designing & Modelling
3.
Testing of single-encoding schemes
Testing of multi-encoding schemes
i.e.. The Magic Mode
5.
Black box testing
4.
6.
Decryption of the cipher text

- **Packages imported in our tool:**

1. Platform:

The Platform module is used to retrieve as much possible information about the platform on which the program is being currently executed

2. argparse

This module argparse help us to create the program in a command line environment that improves our interaction. This module automatically generates help and usage messages and it issues error in case of wrong argument.

3. json

Python comes with a built-in package called json for encoding and decoding JSON data

4. coloroma

A simple cross-platform API for printing colored text from Python

5.termcolor

A python module for ANSII Color formatting for output in the terminal.

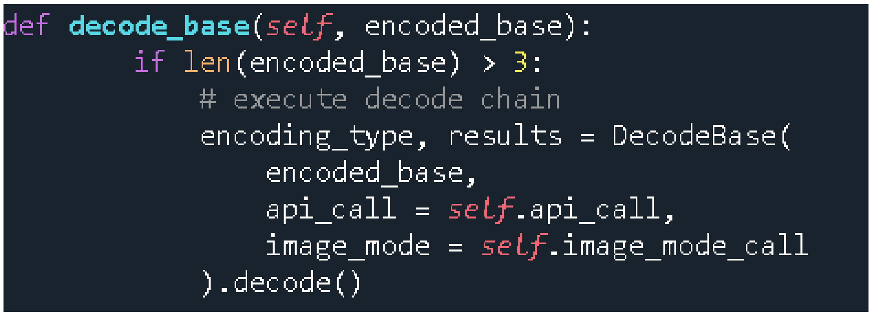
simple cross-platform API to print colored terminal text from Python applications

6.pathlib

Pathlib module provides various classes that represent file system.

- ➢ **<u>Important Functions used in our tool:</u>**

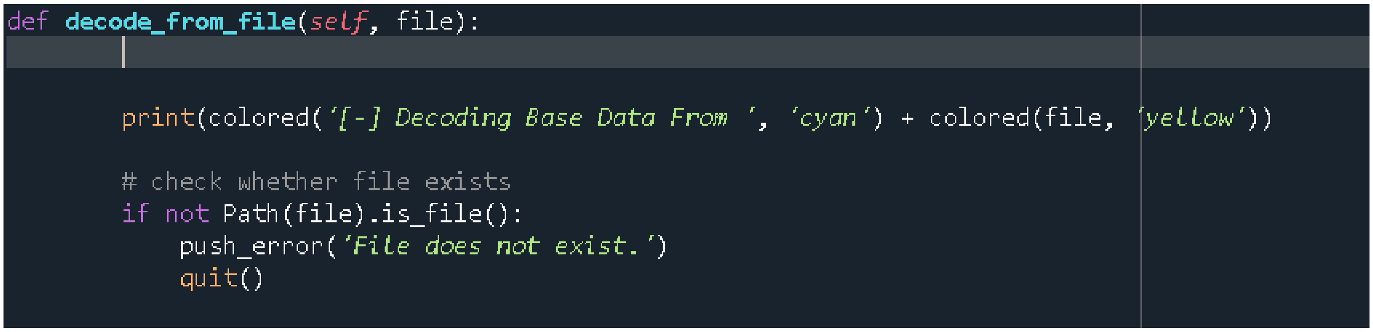
1. **def decode base (self, encoded base):**



```
def decode_base(self, encoded_base):
        if len(encoded_base) > 3:
            # execute decode chain
            encoding_type, results = DecodeBase(
                encoded_base,
                api_call = self.api_call,
                image_mode = self.image_mode_call
            ).decode()
```

This is one of the main functions of our tool since it predicts the heuristics of our cipher text and decode the encoding scheme.

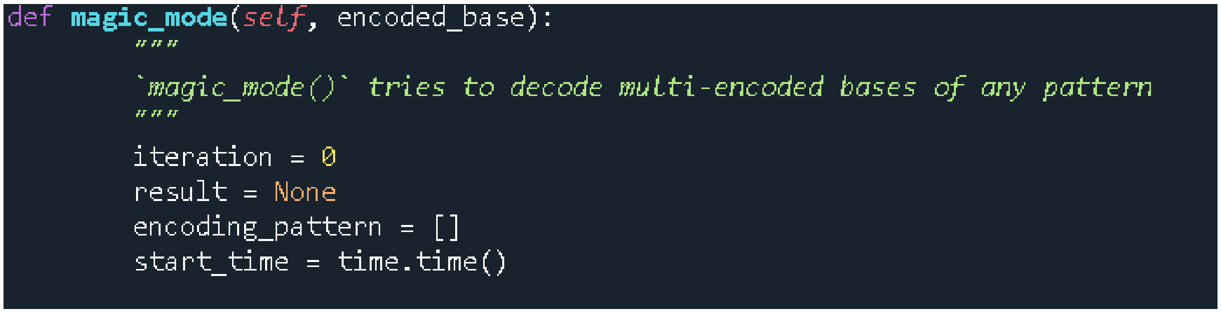
2. **def decode from file (self, file):**



```
def decode_from_file(self, file):


       print(colored('[-] Decoding Base Data From ', 'cyan') + colored(file, 'yellow'))

       # check whether file exists
       if not Path(file).is_file():
           push_error('File does not exist.')
           quit()
```

This function helps to decode an encrypted text present in a file.

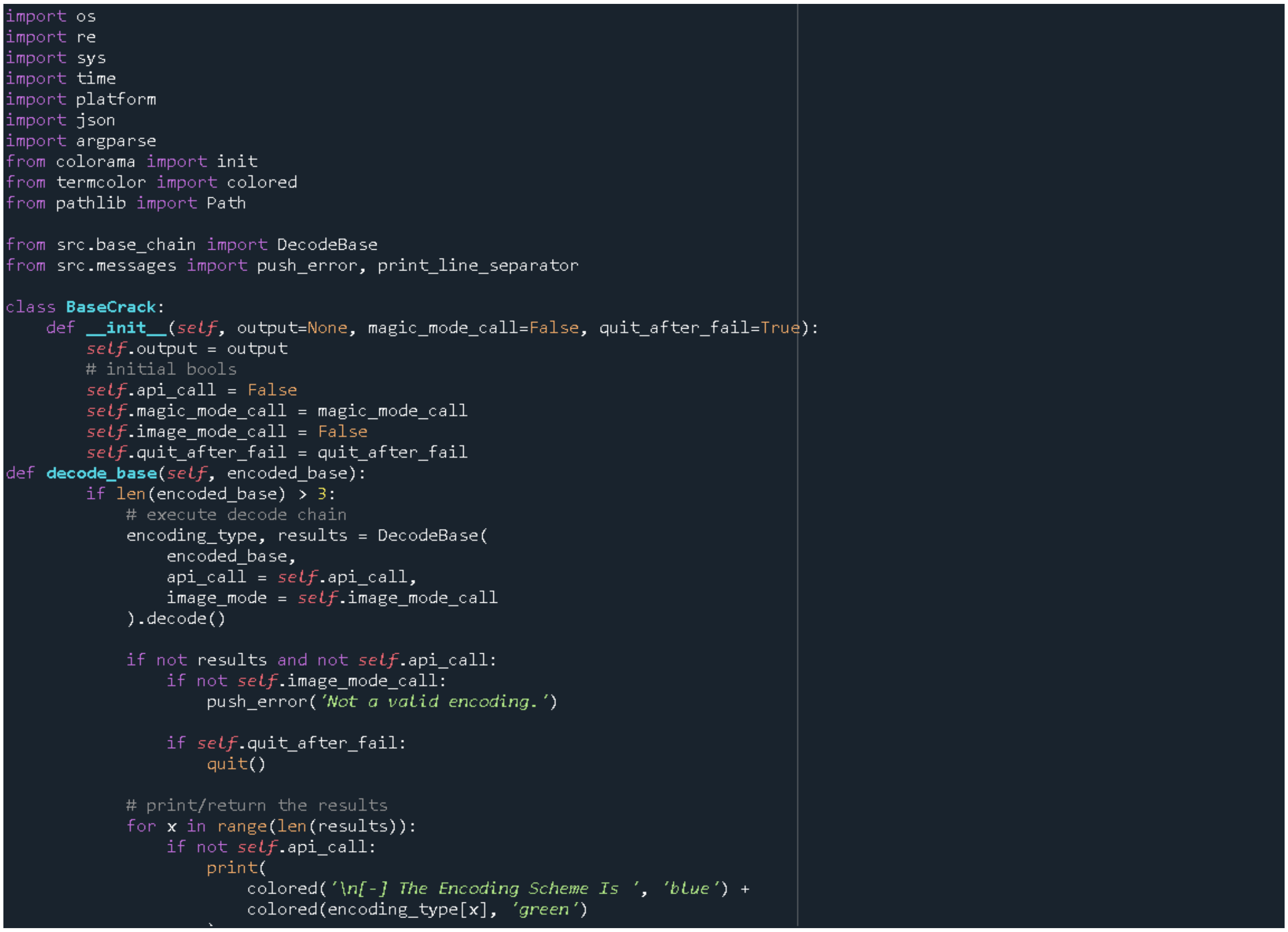
3. **def magic mode (self, encoded base):**



```
def magic_mode(self, encoded_base):
        """
        `magic_mode()` tries to decode multi-encoded bases of any pattern
        """
        iteration = 0
        result = None
        encoding_pattern = []
        start_time = time.time()
```

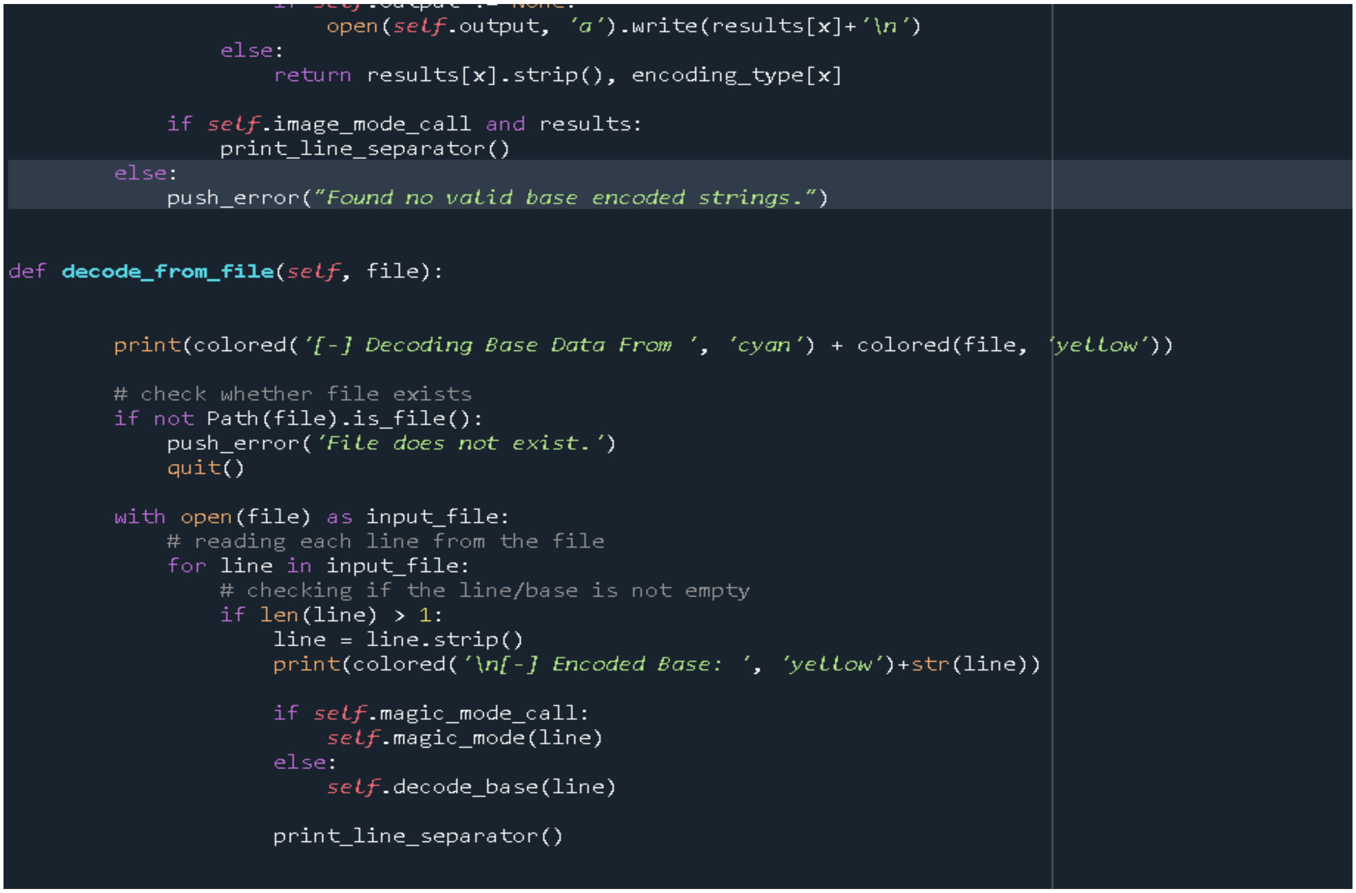
d. Algorithm/Code:

1. Packages/important functions



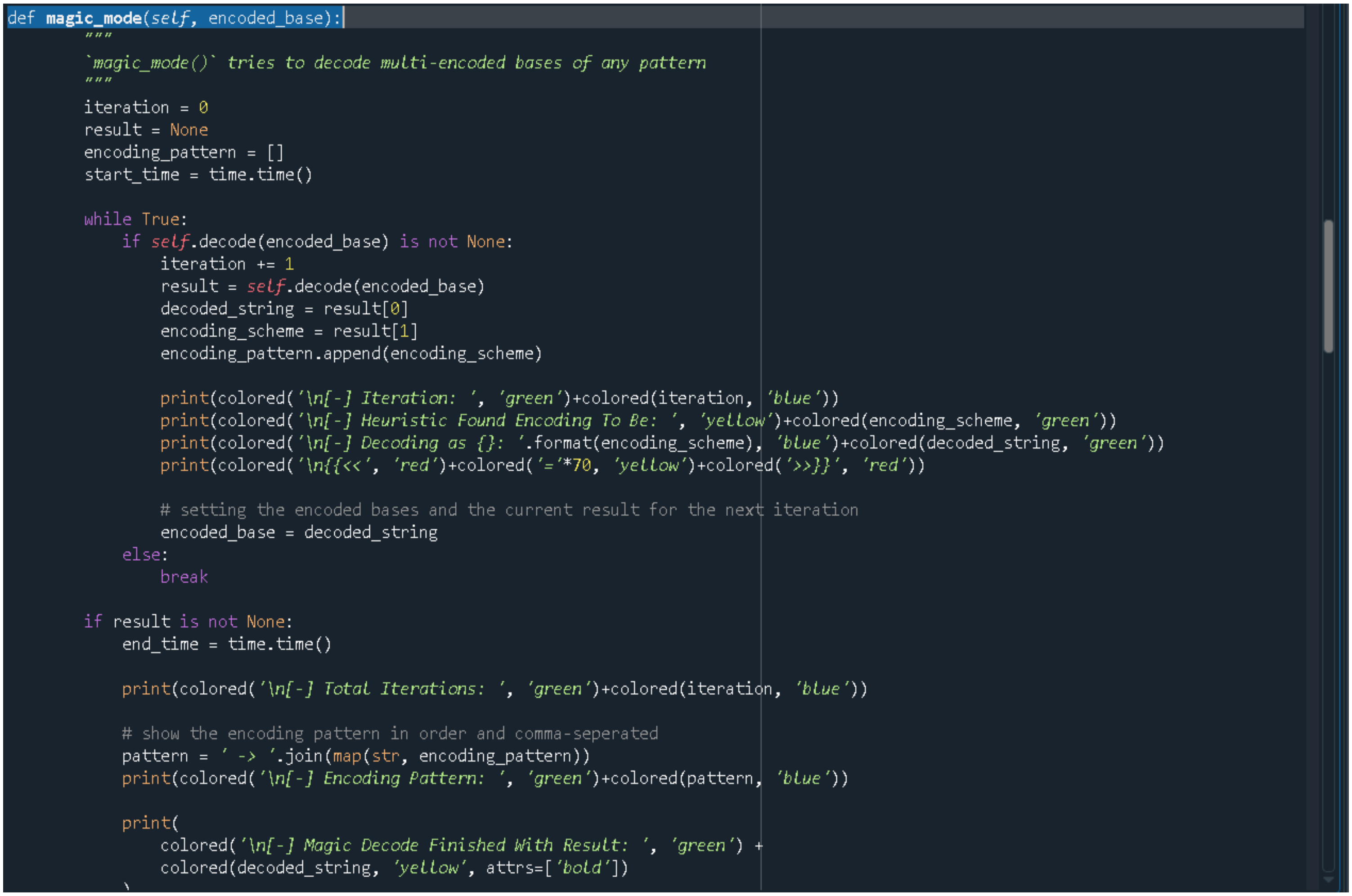
```
import os
import re
import sys
import time
import platform
import json
import argparse
from colorama import init
from termcolor import colored
from pathlib import Path

from src.base_chain import DecodeBase
from src.messages import push_error, print_line_separator

class BaseCrack:
    def __init__(self, output=None, magic_mode_call=False, quit_after_fail=True):
        self.output = output
        # initial bools
        self.api_call = False
        self.magic_mode_call = magic_mode_call
        self.image_mode_call = False
        self.quit_after_fail = quit_after_fail
def decode_base(self, encoded_base):
        if len(encoded_base) > 3:
            # execute decode chain
            encoding_type, results = DecodeBase(
                encoded_base,
                api_call = self.api_call,
                image_mode = self.image_mode_call
            ).decode()

            if not results and not self.api_call:
                if not self.image_mode_call:
                    push_error('Not a valid encoding.')

                if self.quit_after_fail:
                    quit()

            # print/return the results
            for x in range(len(results)):
                if not self.api_call:
                    print(
                        colored('\n[-] The Encoding Scheme Is ', 'blue') +
                        colored(encoding_type[x], 'green')
                    )
```

2. Decode from file function



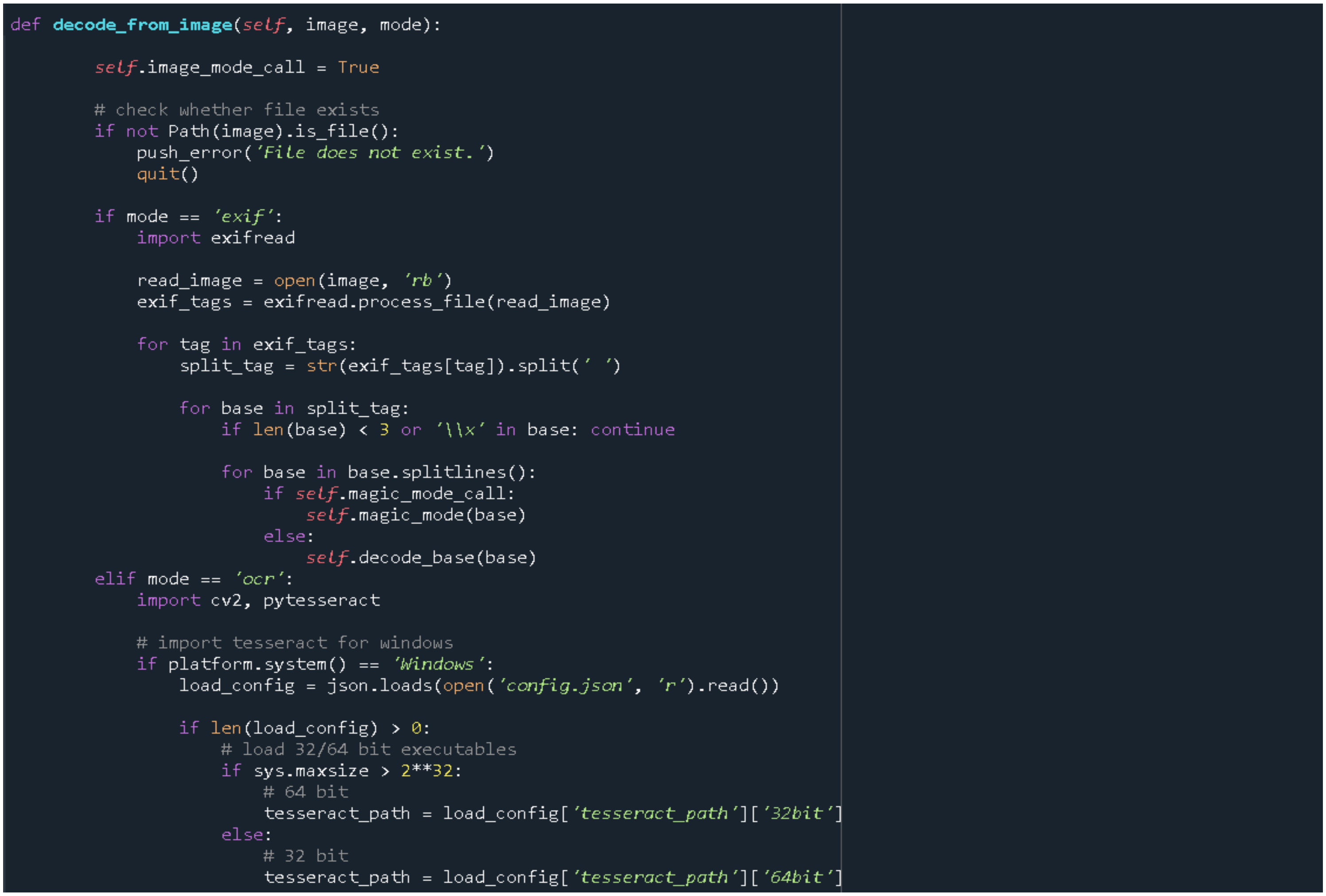
```
                    if self.output != None:
                        open(self.output, 'a').write(results[x]+'\n')
                else:
                    return results[x].strip(), encoding_type[x]

            if self.image_mode_call and results:
                print_line_separator()
        else:
            push_error("Found no valid base encoded strings.")


def decode_from_file(self, file):


        print(colored('[-] Decoding Base Data From ', 'cyan') + colored(file, 'yellow'))

        # check whether file exists
        if not Path(file).is_file():
            push_error('File does not exist.')
            quit()

        with open(file) as input_file:
            # reading each line from the file
            for line in input_file:
                # checking if the line/base is not empty
                if len(line) > 1:
                    line = line.strip()
                    print(colored('\n[-] Encoded Base: ', 'yellow')+str(line))

                    if self.magic_mode_call:
                        self.magic_mode(line)
                    else:
                        self.decode_base(line)

                    print_line_separator()
```

3. Magic mode function



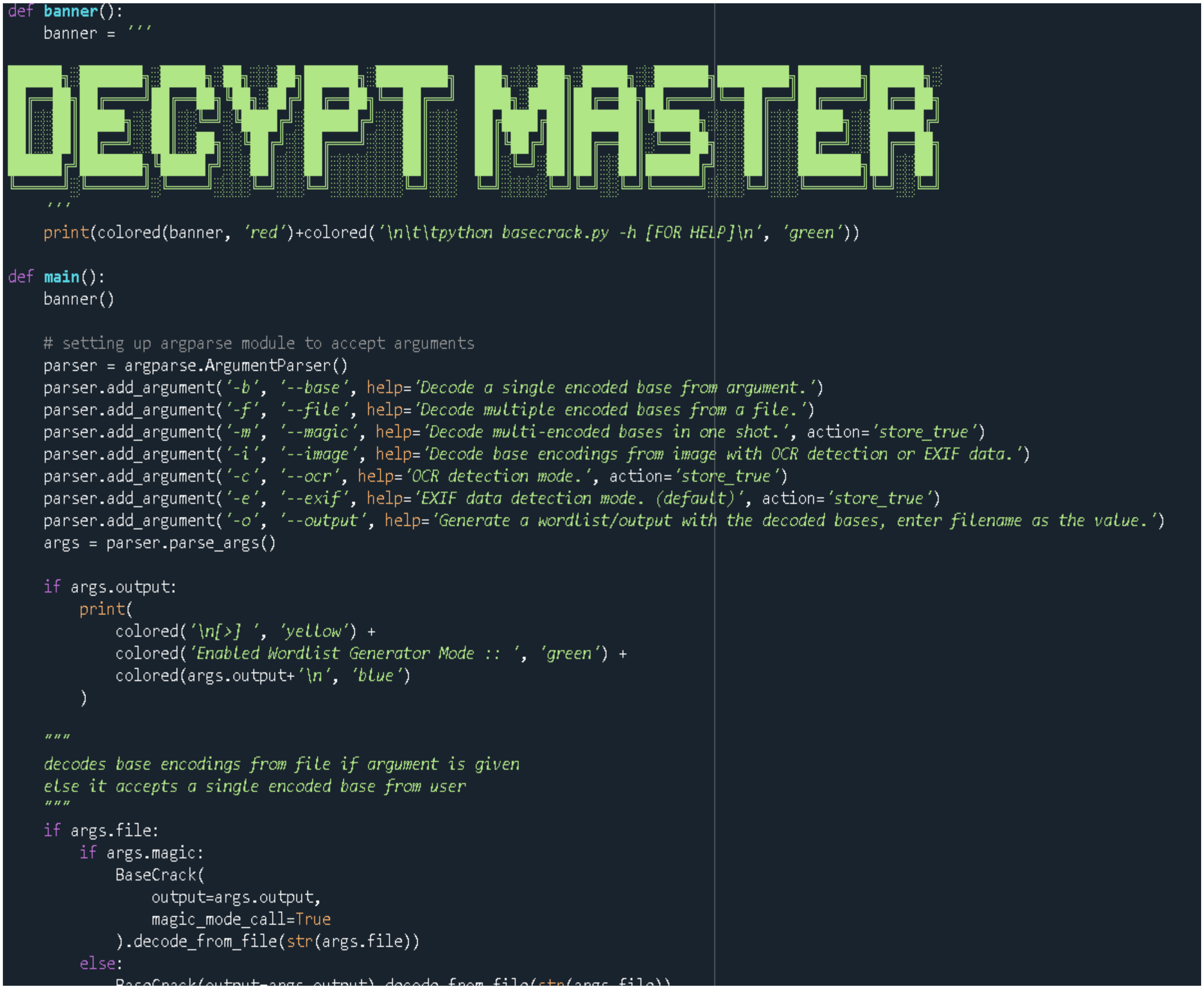
```
def magic_mode(self, encoded_base):
        """
        `magic_mode()` tries to decode multi-encoded bases of any pattern
        """
        iteration = 0
        result = None
        encoding_pattern = []
        start_time = time.time()

        while True:
            if self.decode(encoded_base) is not None:
                iteration += 1
                result = self.decode(encoded_base)
                decoded_string = result[0]
                encoding_scheme = result[1]
                encoding_pattern.append(encoding_scheme)

                print(colored('\n[-] Iteration: ', 'green')+colored(iteration, 'blue'))
                print(colored('\n[-] Heuristic Found Encoding To Be: ', 'yellow')+colored(encoding_scheme, 'green'))
                print(colored('\n[-] Decoding as {}: '.format(encoding_scheme), 'blue')+colored(decoded_string, 'green'))
                print(colored('\n{{<<', 'red')+colored('='*70, 'yellow')+colored('>>}}', 'red'))

                # setting the encoded bases and the current result for the next iteration
                encoded_base = decoded_string
            else:
                break

        if result is not None:
            end_time = time.time()

            print(colored('\n[-] Total Iterations: ', 'green')+colored(iteration, 'blue'))

            # show the encoding pattern in order and comma-seperated
            pattern = ' -> '.join(map(str, encoding_pattern))
            print(colored('\n[-] Encoding Pattern: ', 'green')+colored(pattern, 'blue'))

            print(
                colored('\n[-] Magic Decode Finished With Result: ', 'green') +
                colored(decoded_string, 'yellow', attrs=['bold'])
            )
```

4. Decode from image function



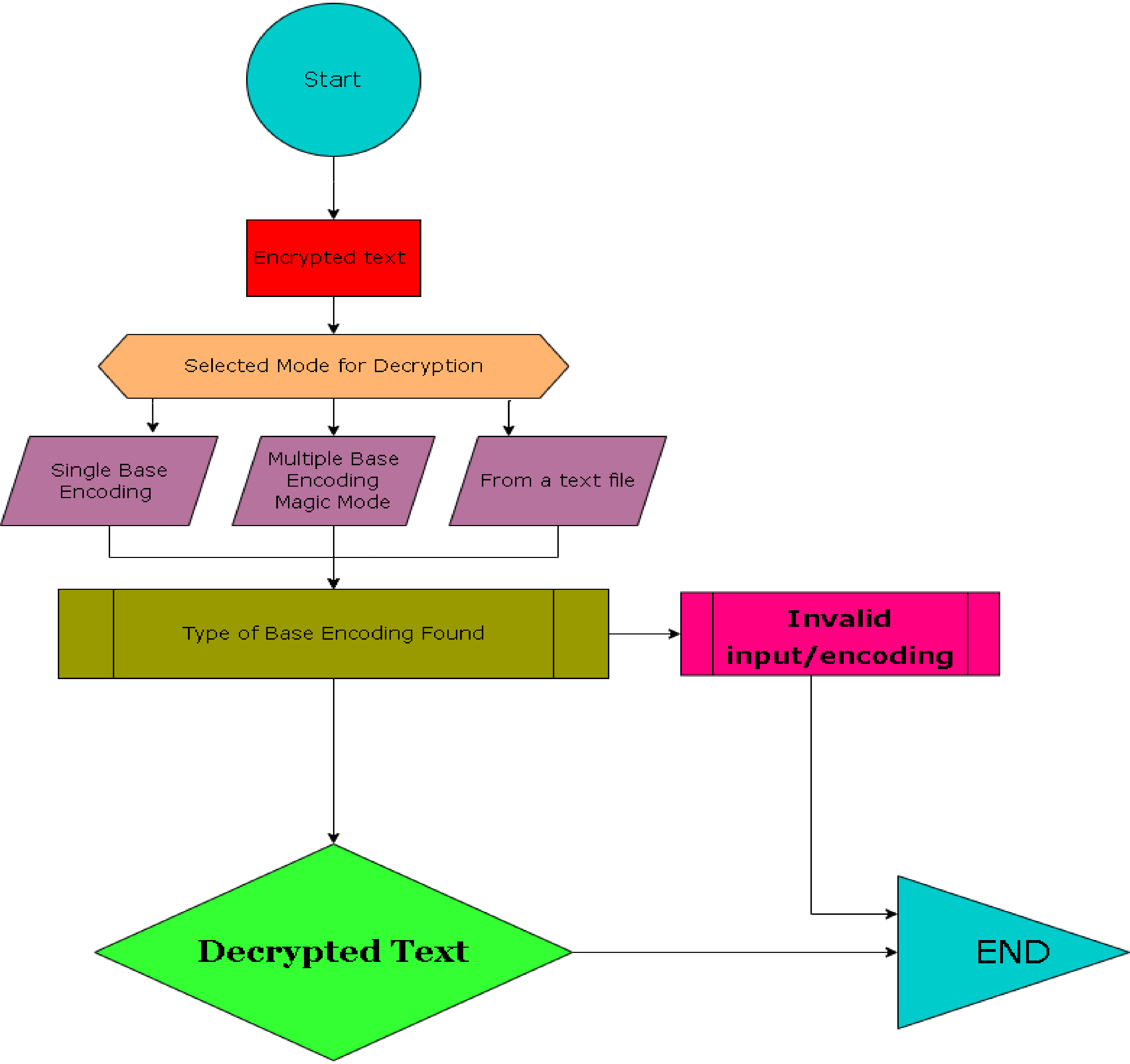
```
def decode_from_image(self, image, mode):

        self.image_mode_call = True

        # check whether file exists
        if not Path(image).is_file():
            push_error('File does not exist.')
            quit()

        if mode == 'exif':
            import exifread

            read_image = open(image, 'rb')
            exif_tags = exifread.process_file(read_image)

            for tag in exif_tags:
                split_tag = str(exif_tags[tag]).split(' ')

                for base in split_tag:
                    if len(base) < 3 or '\\x' in base: continue

                    for base in base.splitlines():
                        if self.magic_mode_call:
                            self.magic_mode(base)
                        else:
                            self.decode_base(base)
        elif mode == 'ocr':
            import cv2, pytesseract

            # import tesseract for windows
            if platform.system() == 'Windows':
                load_config = json.loads(open('config.json', 'r').read())

                if len(load_config) > 0:
                    # load 32/64 bit executables
                    if sys.maxsize > 2**32:
                        # 64 bit
                        tesseract_path = load_config['tesseract_path']['32bit']
                    else:
                        # 32 bit
                        tesseract_path = load_config['tesseract_path']['64bit']
```

5. Banner of the tool



```
def banner():
    banner = '''
DECYPT MASTER
    '''
    print(colored(banner, 'red')+colored('\n\t\tpython basecrack.py -h [FOR HELP]\n', 'green'))

def main():
    banner()

    # setting up argparse module to accept arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('-b', '--base', help='Decode a single encoded base from argument.')
    parser.add_argument('-f', '--file', help='Decode multiple encoded bases from a file.')
    parser.add_argument('-m', '--magic', help='Decode multi-encoded bases in one shot.', action='store_true')
    parser.add_argument('-i', '--image', help='Decode base encodings from image with OCR detection or EXIF data.')
    parser.add_argument('-c', '--ocr', help='OCR detection mode.', action='store_true')
    parser.add_argument('-e', '--exif', help='EXIF data detection mode. (default)', action='store_true')
    parser.add_argument('-o', '--output', help='Generate a wordlist/output with the decoded bases, enter filename as the value.')
    args = parser.parse_args()

    if args.output:
        print(
            colored('\n[>] ', 'yellow') +
            colored('Enabled Wordlist Generator Mode :: ', 'green') +
            colored(args.output+'\n', 'blue')
        )

    """
    decodes base encodings from file if argument is given
    else it accepts a single encoded base from user
    """
    if args.file:
        if args.magic:
            BaseCrack(
                output=args.output,
                magic_mode_call=True
            ).decode_from_file(str(args.file))
        else:
            BaseCrack(output=args.output).decode_from_file(str(args.file))
```

e. Flow chart:



4. Performance Analysis

After carefully designing, modelling our tool, we achieved our final objectives. Decrypt Master works just fine, it decodes single encoded base, multi-encoded base and base from a text file.

Observation

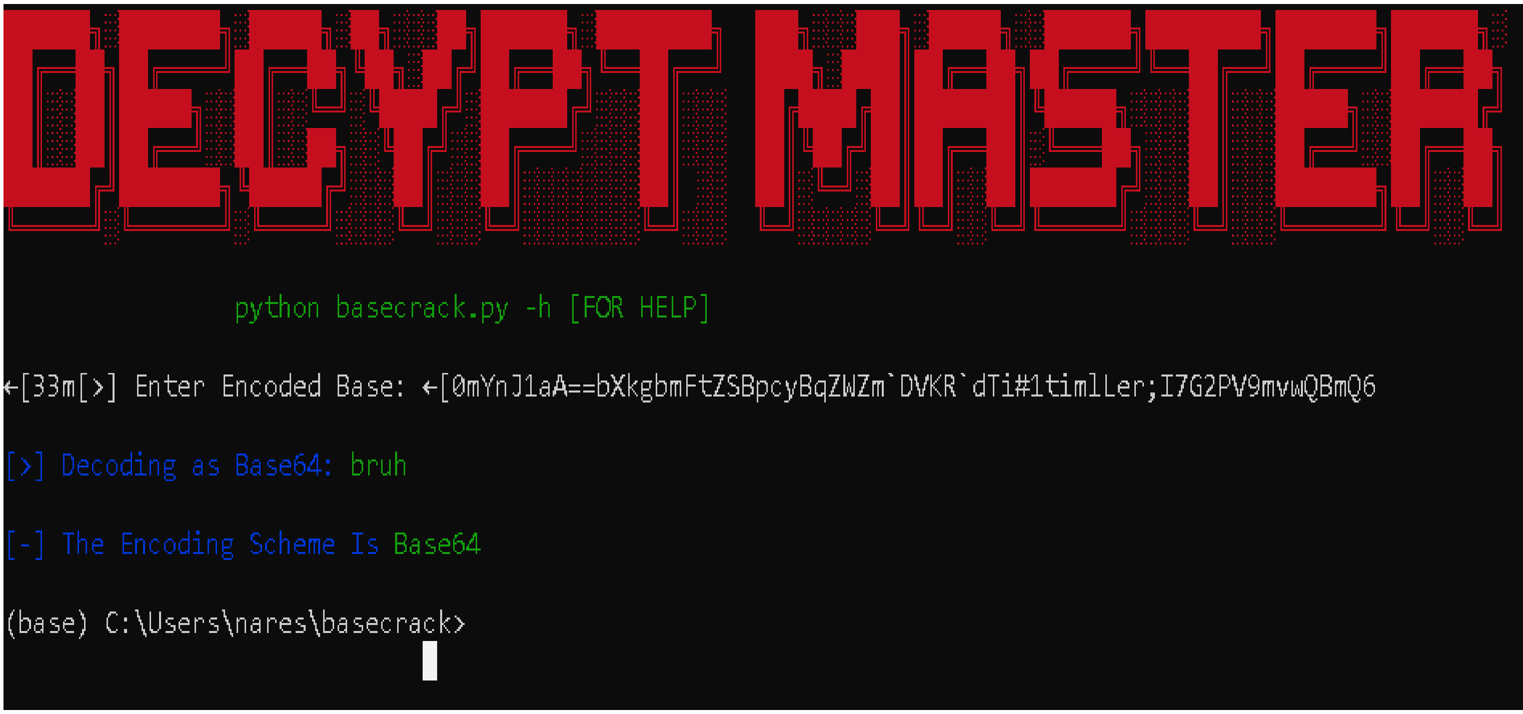
Snaps of the tool after various types of input and output:

1. **Base64 encoded text:**

Encrypted text:

YnJ1aA==bXkgbmFtZSBpcyBqZWZm`DVKR`dTi#1timlLer;I7G2PV9mvwQBmQ6

Decrypted text:



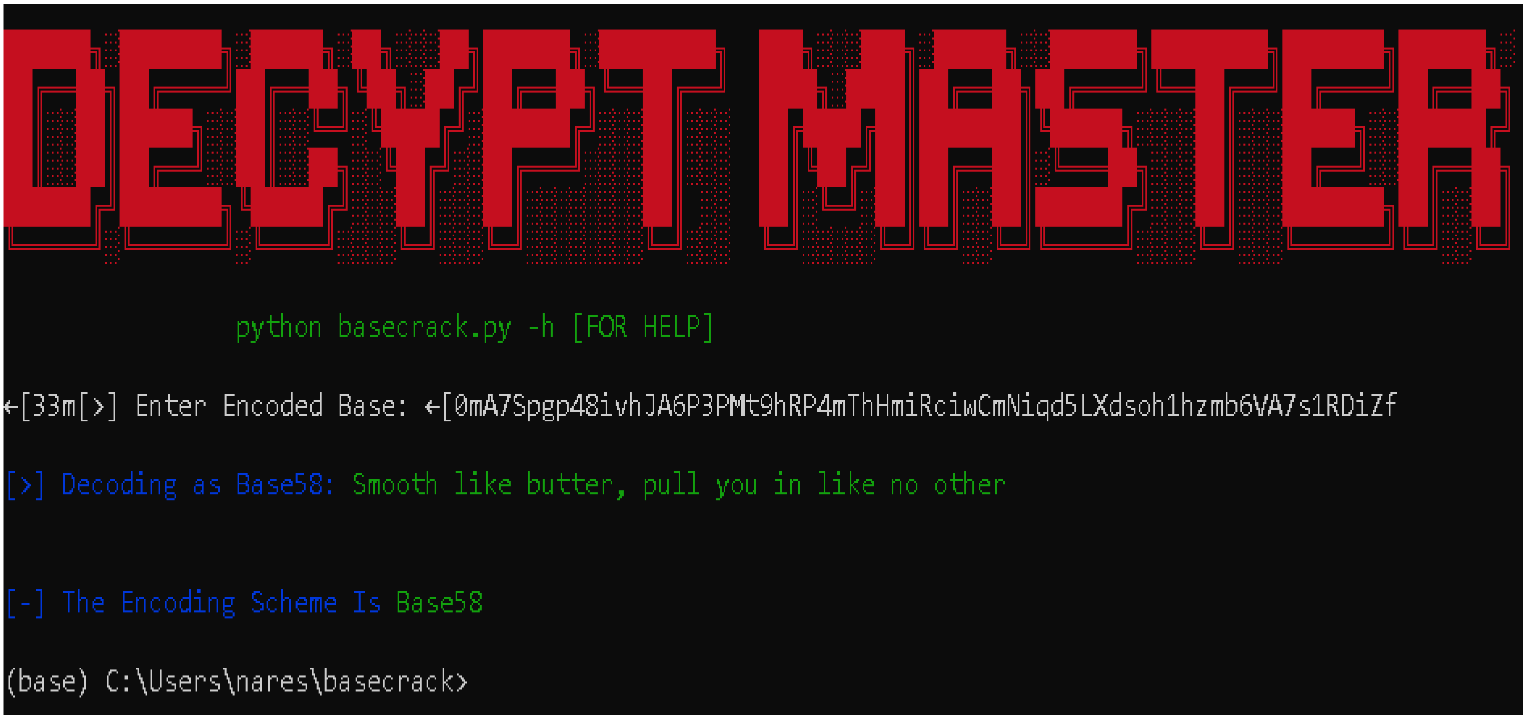
```
                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0mYnJ1aA==bXkgbmFtZSBpcyBqZWZm`DVKR`dTi#1timlLer;I7G2PV9mvwQBmQ6

[>] Decoding as Base64: bruh

[-] The Encoding Scheme Is Base64

(base) C:\Users\nares\basecrack>
```

2. **Base58 encoded text:**

Encrypted text:

A7Spgp48ivhJA6P3PMt9hRP4mThHmiRciwCmNiqd5LXdsoh1hzmb6VA7s1RDiZf

Decrypted text:



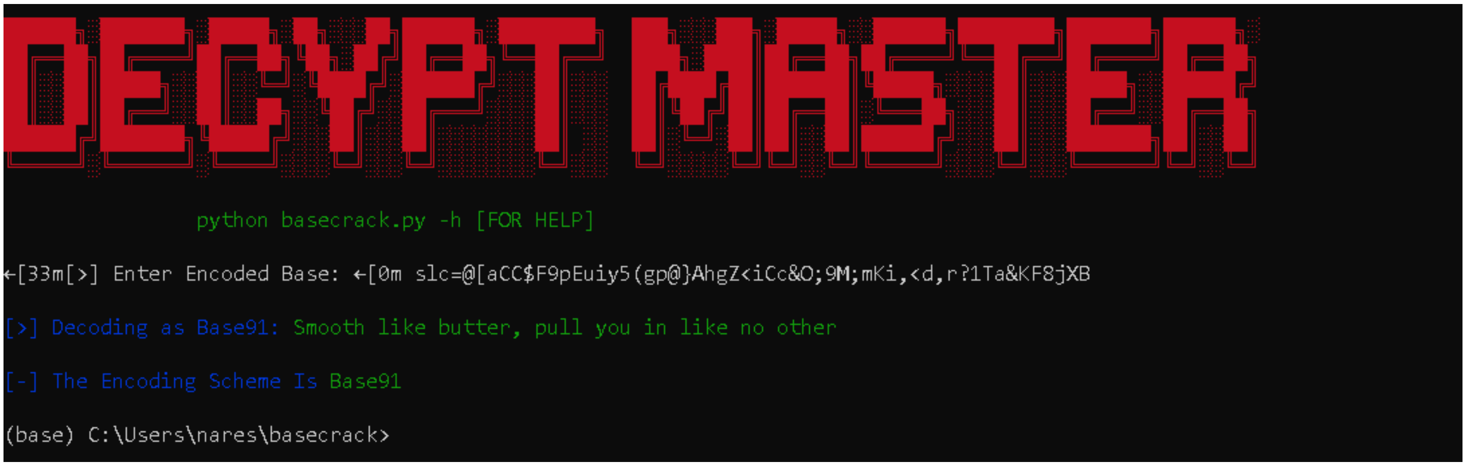
```
                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0mA7Spgp48ivhJA6P3PMt9hRP4mThHmiRciwCmNiqd5LXdsoh1hzmb6VA7s1RDiZf

[>] Decoding as Base58: Smooth like butter, pull you in like no other


[-] The Encoding Scheme Is Base58

(base) C:\Users\nares\basecrack>
```

3. Base91 encoded text:

Encrypted text:

slc=@[aCC$F9pEuiy5(gp@}AhgZ<iCc&O;9M;mKi,<d,r?1Ta&KF8jXB

Decrypted text:



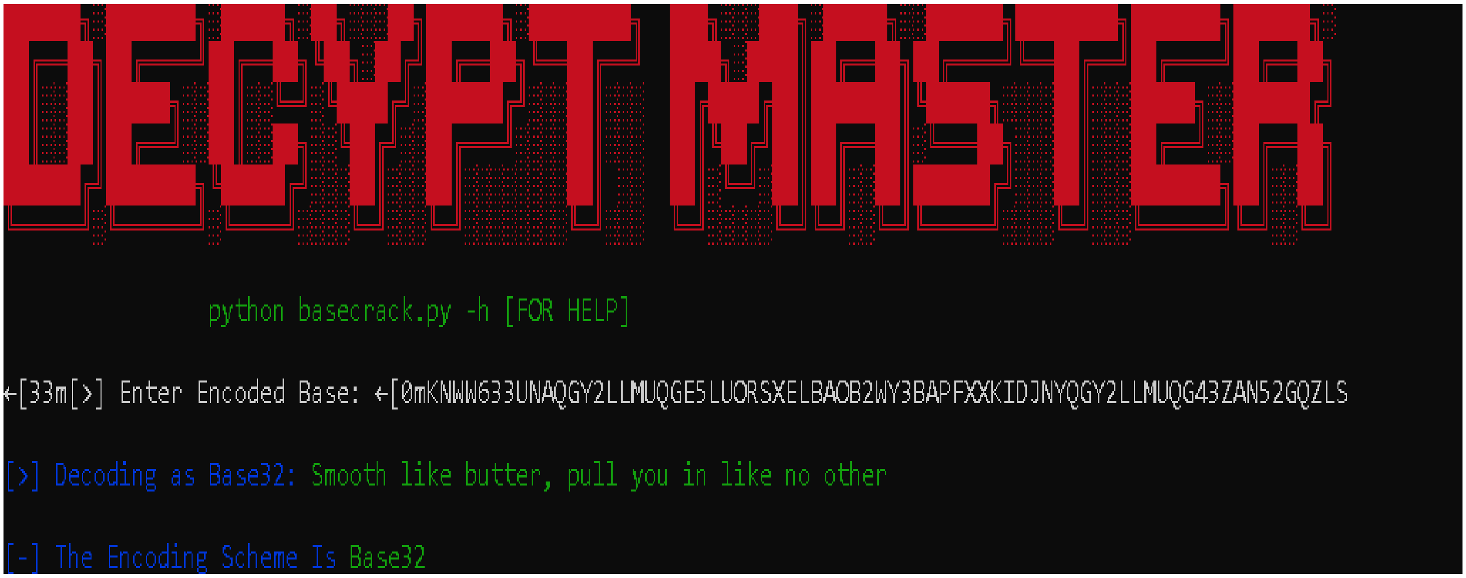
```
                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0m slc=@[aCC$F9pEuiy5(gp@}AhgZ<iCc&O;9M;mKi,<d,r?1Ta&KF8jXB

[>] Decoding as Base91: Smooth like butter, pull you in like no other

[-] The Encoding Scheme Is Base91

(base) C:\Users\nares\basecrack>
```

4. Base32 encoded text:

Encrypted text:

KNWW633UNAQGY2LLMUQGE5LUORSXELBAOB2WY3BAPFXXKIDJNYQGY2LLMUQG43ZAN52GQZLS

Decrypted text:



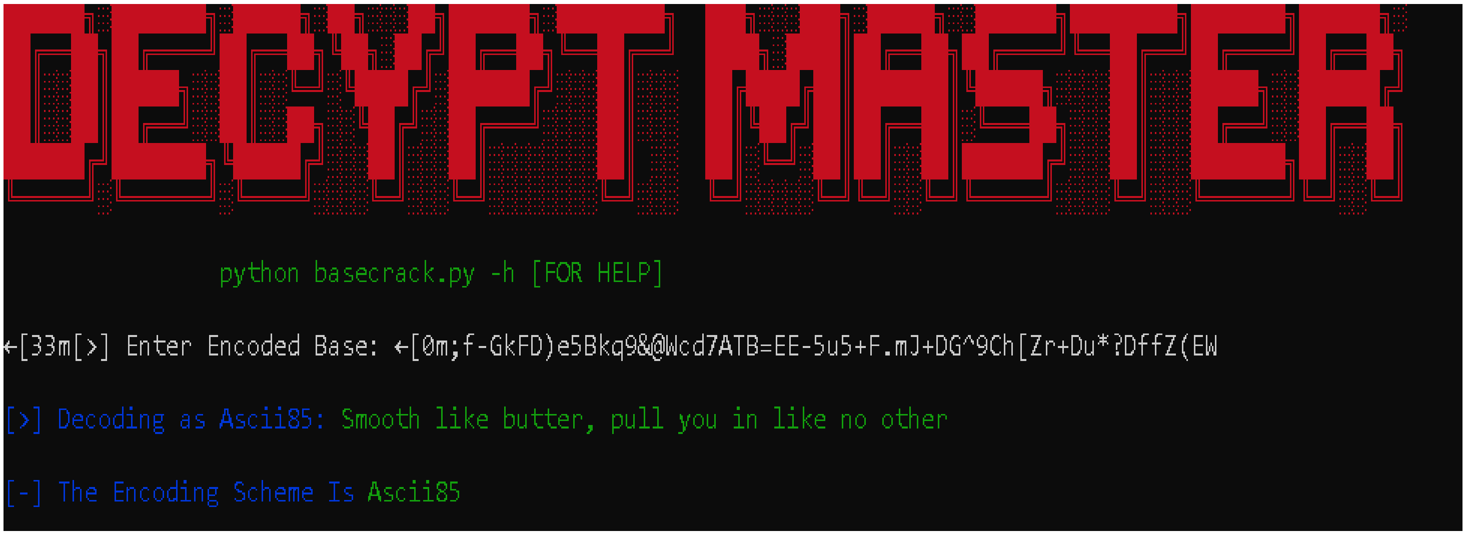
```
                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0mKNWW633UNAQGY2LLMUQGE5LUORSXELBAOB2WY3BAPFXXKIDJNYQGY2LLMUQG43ZAN52GQZLS

[>] Decoding as Base32: Smooth like butter, pull you in like no other

[-] The Encoding Scheme Is Base32
```

5. Base85 encoded text:

Encrypted text:

;f-GkFD)e5Bkq9&@Wcd7ATB=EE-5u5+F.mJ+DG^9Ch[Zr+Du*?DffZ(EW

Decrypted text:



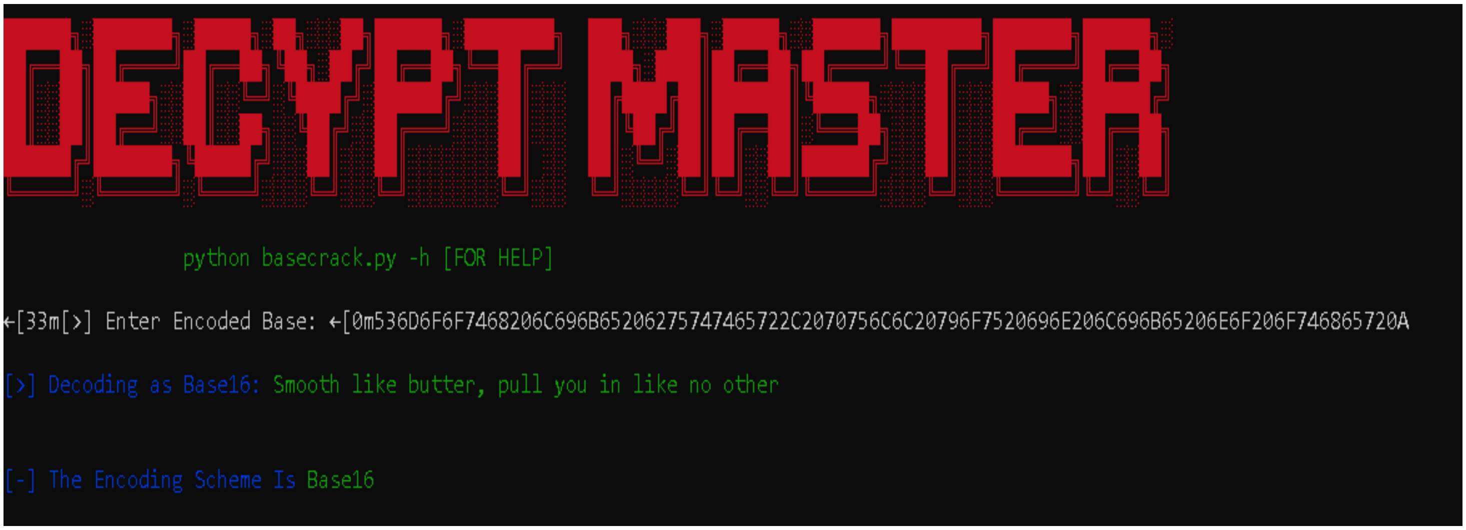
```
                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0m;f-GkFD)e5Bkq9&@Wcd7ATB=EE-5u5+F.mJ+DG^9Ch[Zr+Du*?DffZ(EW

[>] Decoding as Ascii85: Smooth like butter, pull you in like no other

[-] The Encoding Scheme Is Ascii85
```

6. Base16 encoded text:

Encrypted text:

536D6F6F7468206C696B652062757474657222C2070756C6C20796F7520696E206C69 6B65206E6F206F746865720A

Decrypted text:



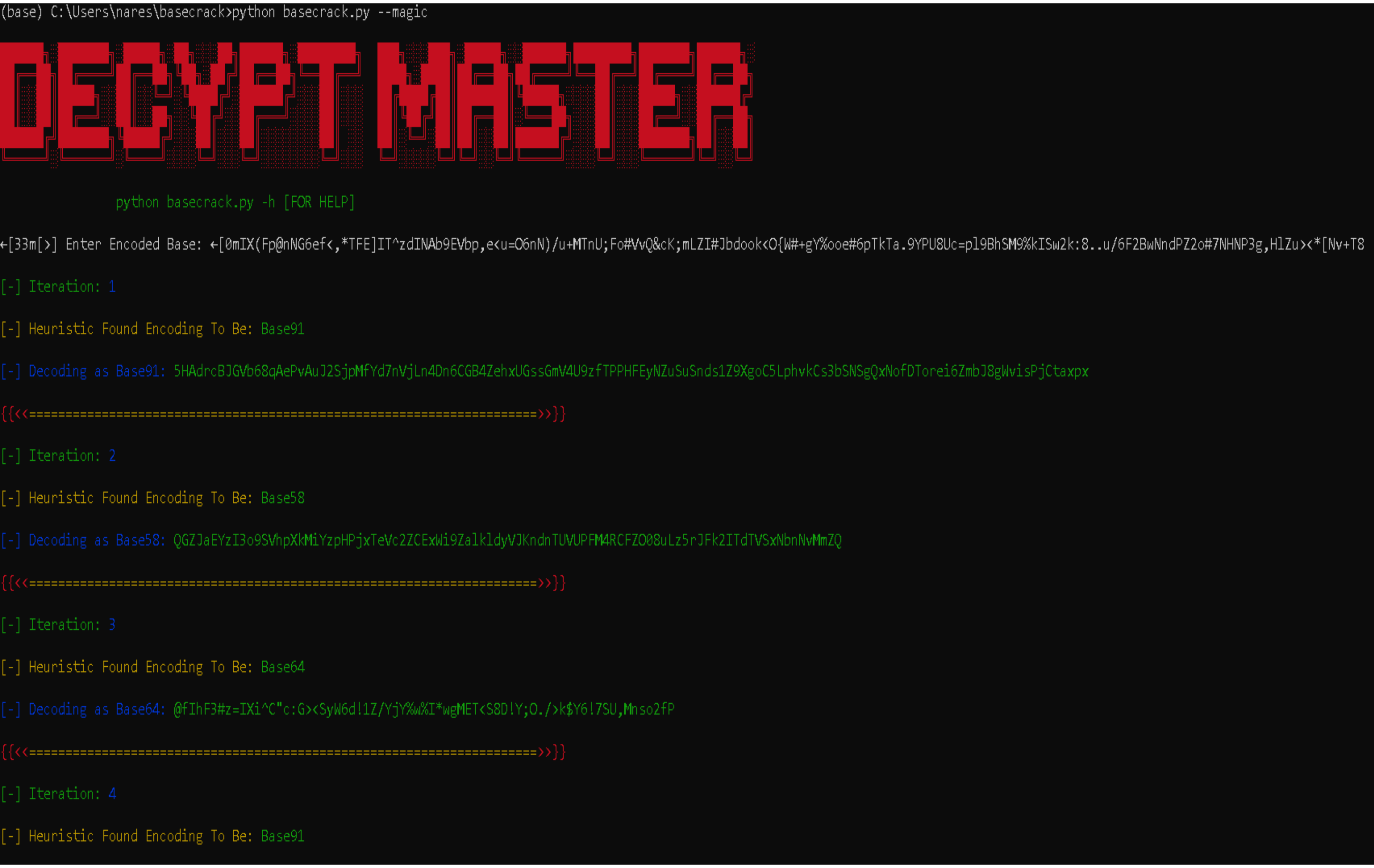
```
            python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0m536D6F6F7468206C696B65206275747465722C2070756C6C20796F7520696E206C696B65206E6F206F746865720A

[>] Decoding as Base16: Smooth like butter, pull you in like no other


[-] The Encoding Scheme Is Base16
```

7. Multiple Encoded text (Magic Mode)

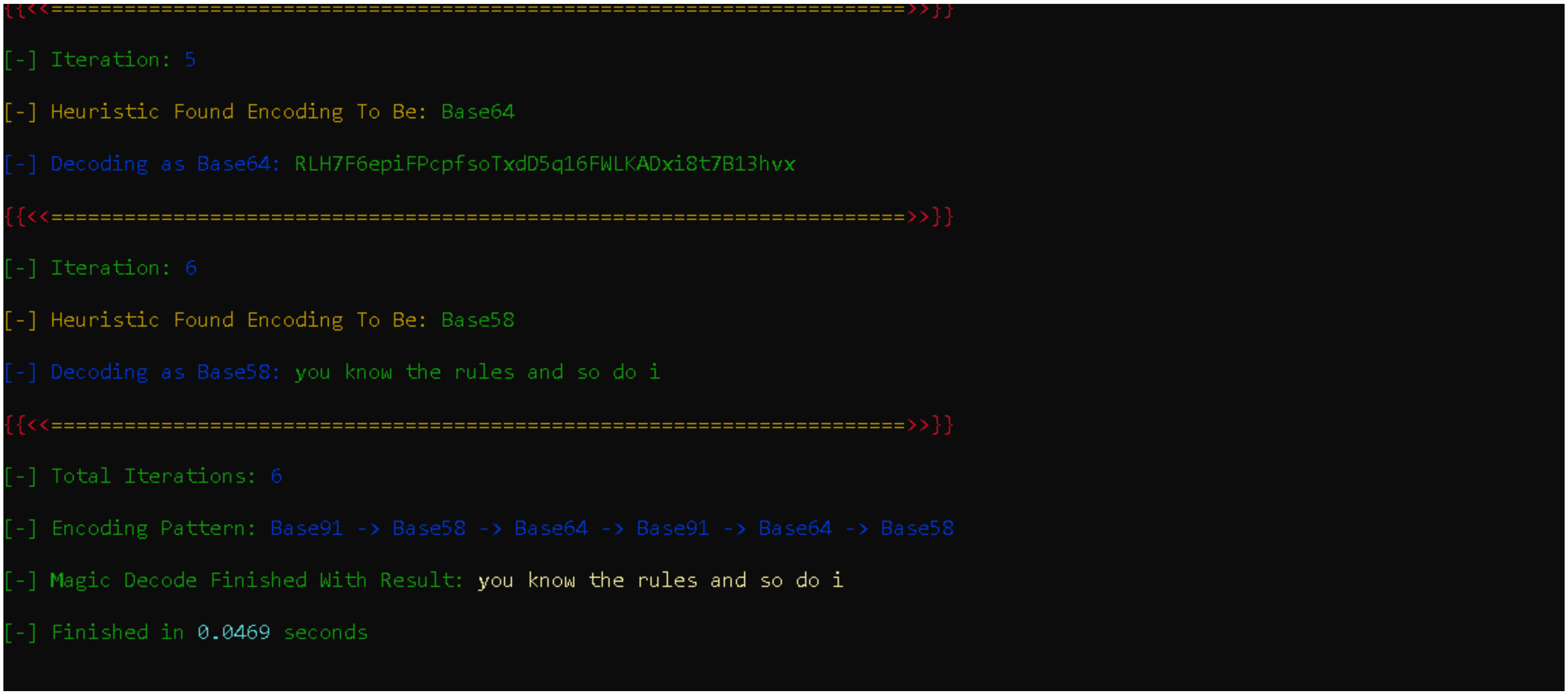
Encrypted text:

```
IX(Fp@nNG6ef<,*TFE]IT^zdINAb9EVbp,e<u=O6nN)/u+MTnU;Fo#VvQ&cK;mLZI#Jbdook<O{W#+gY%ooe#6pTkTa.9YPU8Uc=pl9BhSM9%kISw2k:8..u/6F2BwNndPZ2o#7NHNP3g,HlZu><*[Nv+T8
```

Decrypted text 1:



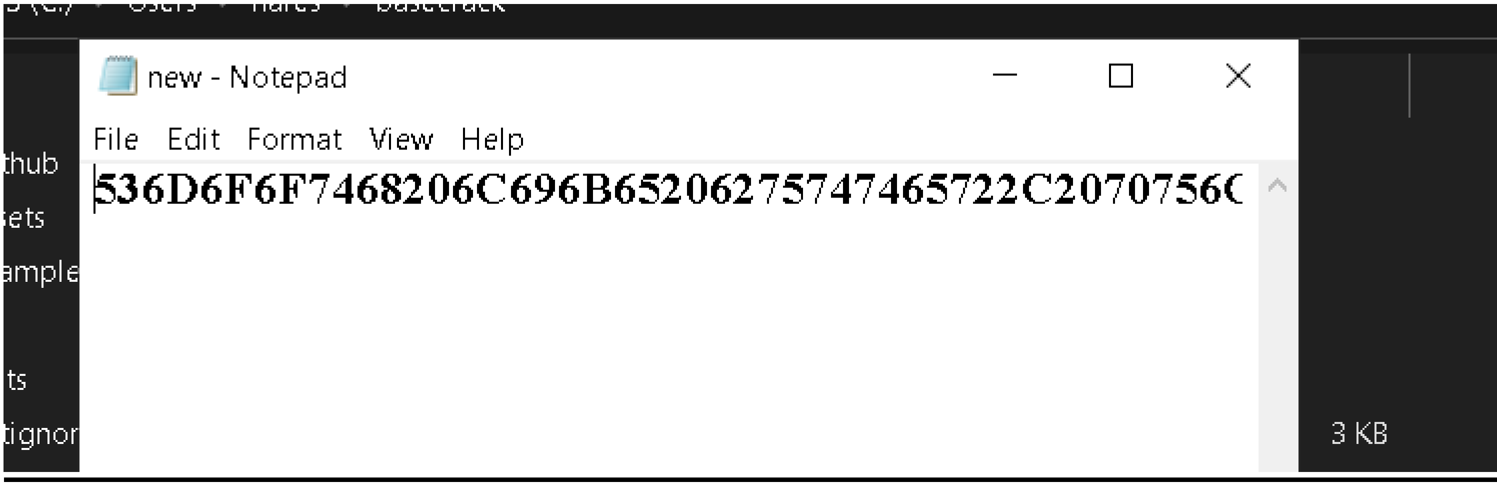
```
(base) C:\Users\nares\basecrack>python basecrack.py --magic

                python basecrack.py -h [FOR HELP]

←[33m[>] Enter Encoded Base: ←[0mIX(Fp@nNG6ef<,*TFE]IT^zdINAb9EVbp,e<u=O6nN)/u+MTnU;Fo#VvQ&cK;mLZI#Jbdook<O{W#+gY%ooe#6pTkTa.9YPU8Uc=pl9BhSM9%kISw2k:8..u/6F2BwNndPZ2o#7NHNP3g,HlZu><*[Nv+T8

[-] Iteration: 1

[-] Heuristic Found Encoding To Be: Base91

[-] Decoding as Base91: 5HAdrcBJGVb68qAePvAuJ2SjpMfYd7nVjLn4Dn6CGB4ZehxUGssGmV4U9zfTPPHFEyNZuSuSnds1Z9XgoC5LphvkCs3bSNSgQxNofDTorei6ZmbJ8gWvisPjCtaxpx

{{<<=========================================================================>>}}

[-] Iteration: 2

[-] Heuristic Found Encoding To Be: Base58

[-] Decoding as Base58: QGZJaEYzI3o9SVhpXkMiYzpHPjxTeVc2ZCExWi9ZalkldyVJKndnTUVUPFM4RCFZO08uLz5rJFk2ITdTVSxNbnNvMmZQ

{{<<=========================================================================>>}}

[-] Iteration: 3

[-] Heuristic Found Encoding To Be: Base64

[-] Decoding as Base64: @fIhF3#z=IXi^C"c:G><SyW6d!1Z/YjY%w%I*wgMET<S8D!Y;O./>k$Y6!7SU,Mnso2fP

{{<<=========================================================================>>}}

[-] Iteration: 4

[-] Heuristic Found Encoding To Be: Base91
```



```
{{<<=========================================================================>>}}

[-] Iteration: 5

[-] Heuristic Found Encoding To Be: Base64

[-] Decoding as Base64: RLH7F6epiFPcpfsoTxdD5q16FWLKADxi8t7B13hvx

{{<<=========================================================================>>}}

[-] Iteration: 6

[-] Heuristic Found Encoding To Be: Base58

[-] Decoding as Base58: you know the rules and so do i

{{<<=========================================================================>>}}

[-] Total Iterations: 6

[-] Encoding Pattern: Base91 -> Base58 -> Base64 -> Base91 -> Base64 -> Base58

[-] Magic Decode Finished With Result: you know the rules and so do i

[-] Finished in 0.0469 seconds
```

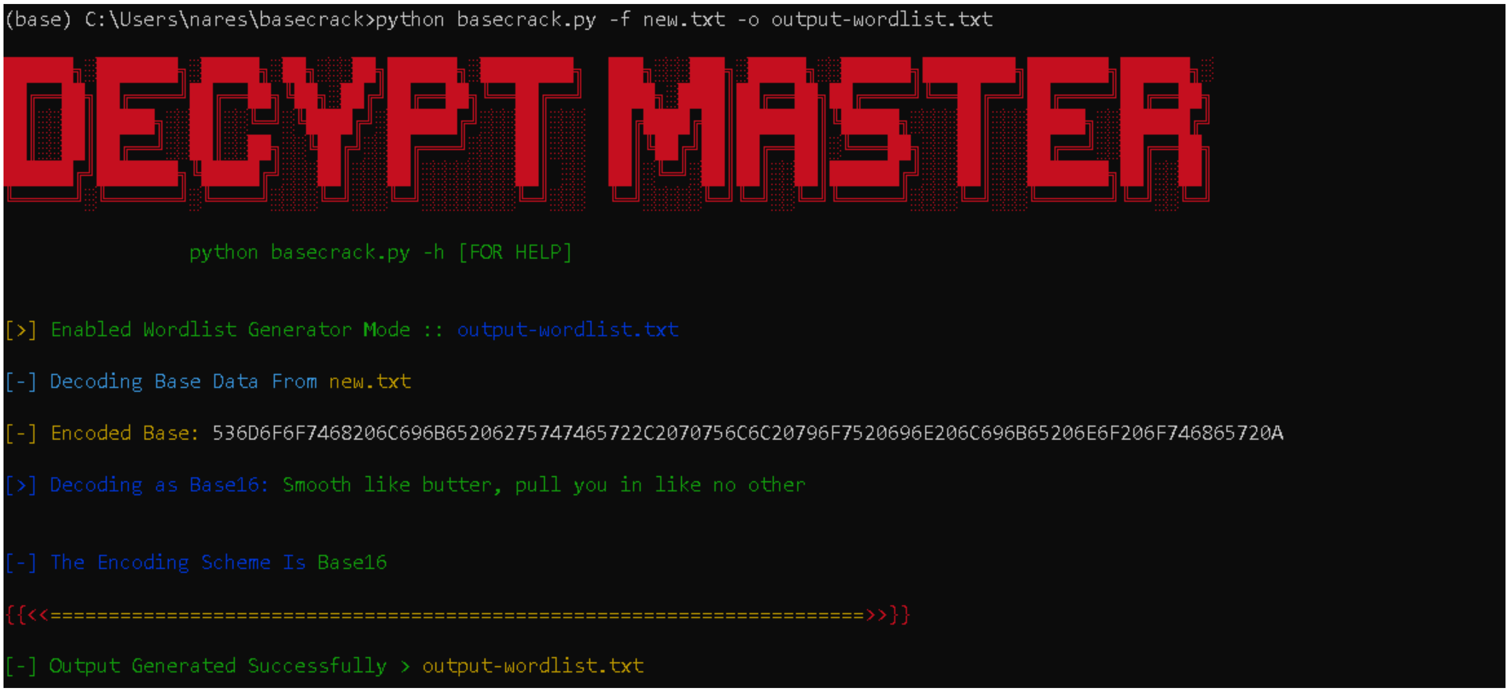
8. From a text file:

Encrypted text:



```
536D6F6F7468206C696B65206275747465722C2070756C
```

Decrypted text:



```
(base) C:\Users\nares\basecrack>python basecrack.py -f new.txt -o output-wordlist.txt

                python basecrack.py -h [FOR HELP]

[>] Enabled Wordlist Generator Mode :: output-wordlist.txt

[-] Decoding Base Data From new.txt

[-] Encoded Base: 536D6F6F7468206C696B65206275747465722C2070756C6C20796F7520696E206C696B65206E6F206F746865720A

[>] Decoding as Base16: Smooth like butter, pull you in like no other


[-] The Encoding Scheme Is Base16

{{<<======================================================================>>}}

[-] Output Generated Successfully > output-wordlist.txt
```

Black-Box Testing

Black-box testing is an essential part of our performance analysis since it will help us find loop holes in our tool and will help us improve its functionality as well as features.

TEST CASE	EXPECTED OUTCOME	OUTCOME OBSERVED	STATUS

Base64,91 encryption	Base64	Base64	Fail
Base91 encryption	Base91	Base91	Pass
00000	Error	Error	Pass
Base36 encryption	Base36	Base36	Pass
Xxxx000000	Not valid	Not valid	Pass
Base64,85,91 encryption	Base64 ,85,91	Base64 ,85,91	Pass

5. Conclusion

- Modern computing technology has made it practical to use far more complex encryption algorithms that are harder to “break” by cryptanalysts.
- In parallel, cryptanalysts have adopted and developed this technology to improve their ability to break cryptosystems
- Security has become a top priority for everyone in I.T these days.
- A report from the Gartner forecasts worldwide security predicts that the total cost for information security and risk management will exceeded $150 billion.

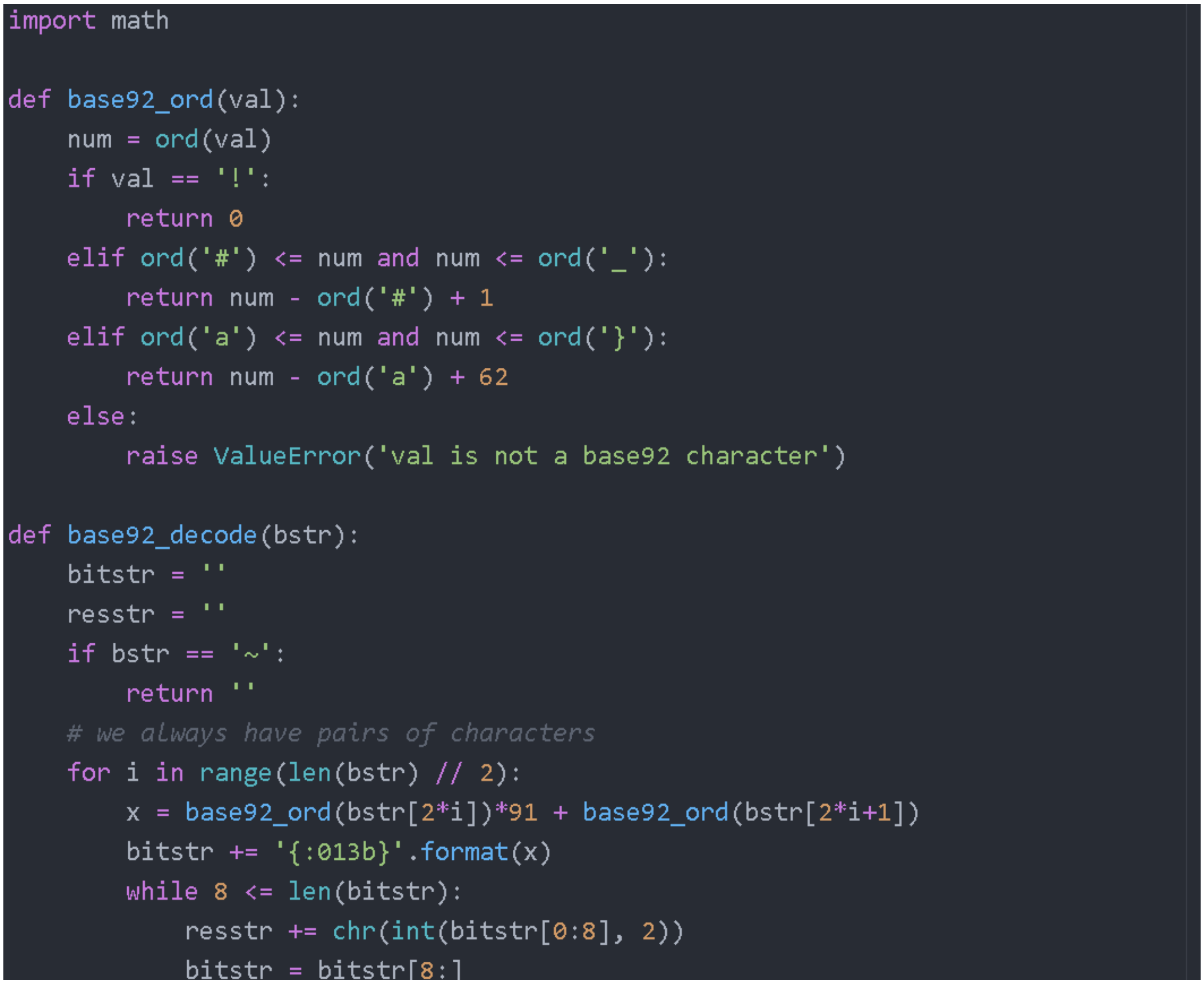
Our tool Decrypt master has been successful in term of achieving its functional and non- functional requirements.

By this tool it would be easier for any cryptographer to decode any base encoding ciphered text.

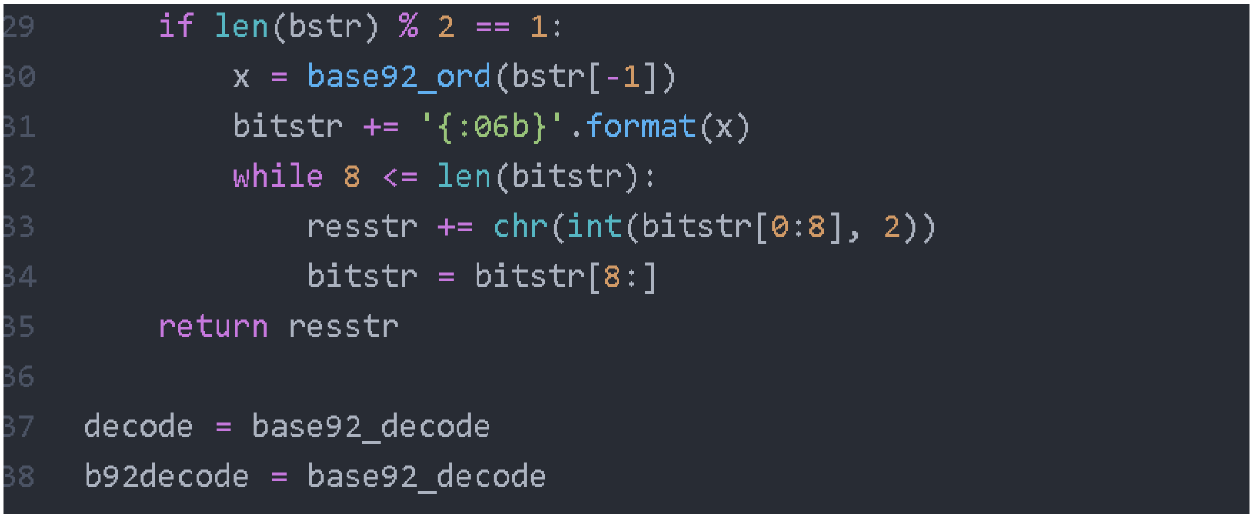
There are still a lot of methods that we can use to secure our data encryption is one aspect of security technology that ever IT enthusiast should understand.

5. Code Snapshots

- **Base 92**

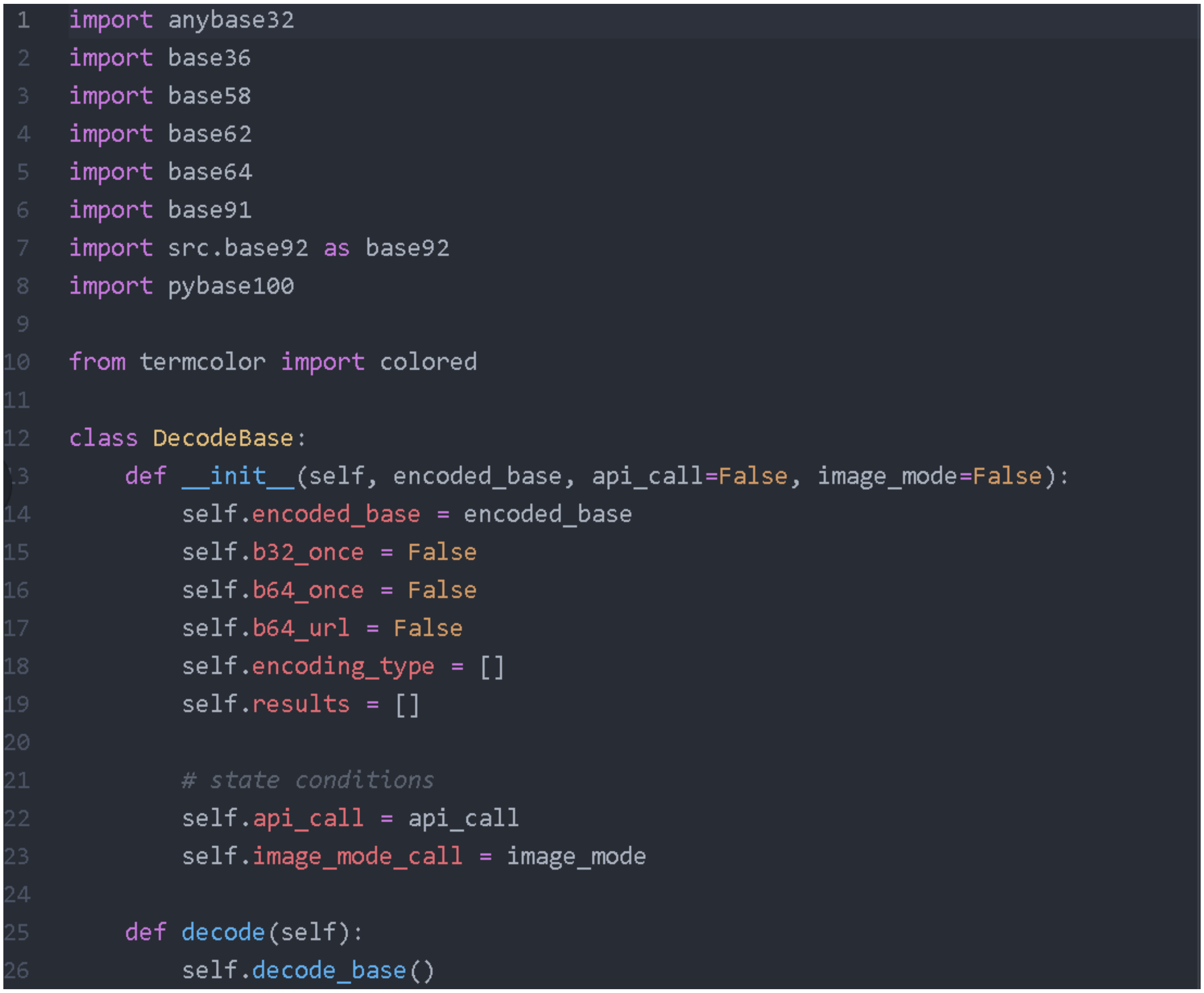


```
import math

def base92_ord(val):
    num = ord(val)
    if val == '!':
        return 0
    elif ord('#') <= num and num <= ord('_'):
        return num - ord('#') + 1
    elif ord('a') <= num and num <= ord('}'):
        return num - ord('a') + 62
    else:
        raise ValueError('val is not a base92 character')

def base92_decode(bstr):
    bitstr = ''
    resstr = ''
    if bstr == '~':
        return ''
    # we always have pairs of characters
    for i in range(len(bstr) // 2):
        x = base92_ord(bstr[2*i])*91 + base92_ord(bstr[2*i+1])
        bitstr += '{:013b}'.format(x)
        while 8 <= len(bitstr):
            resstr += chr(int(bitstr[0:8], 2))
            bitstr = bitstr[8:]
```

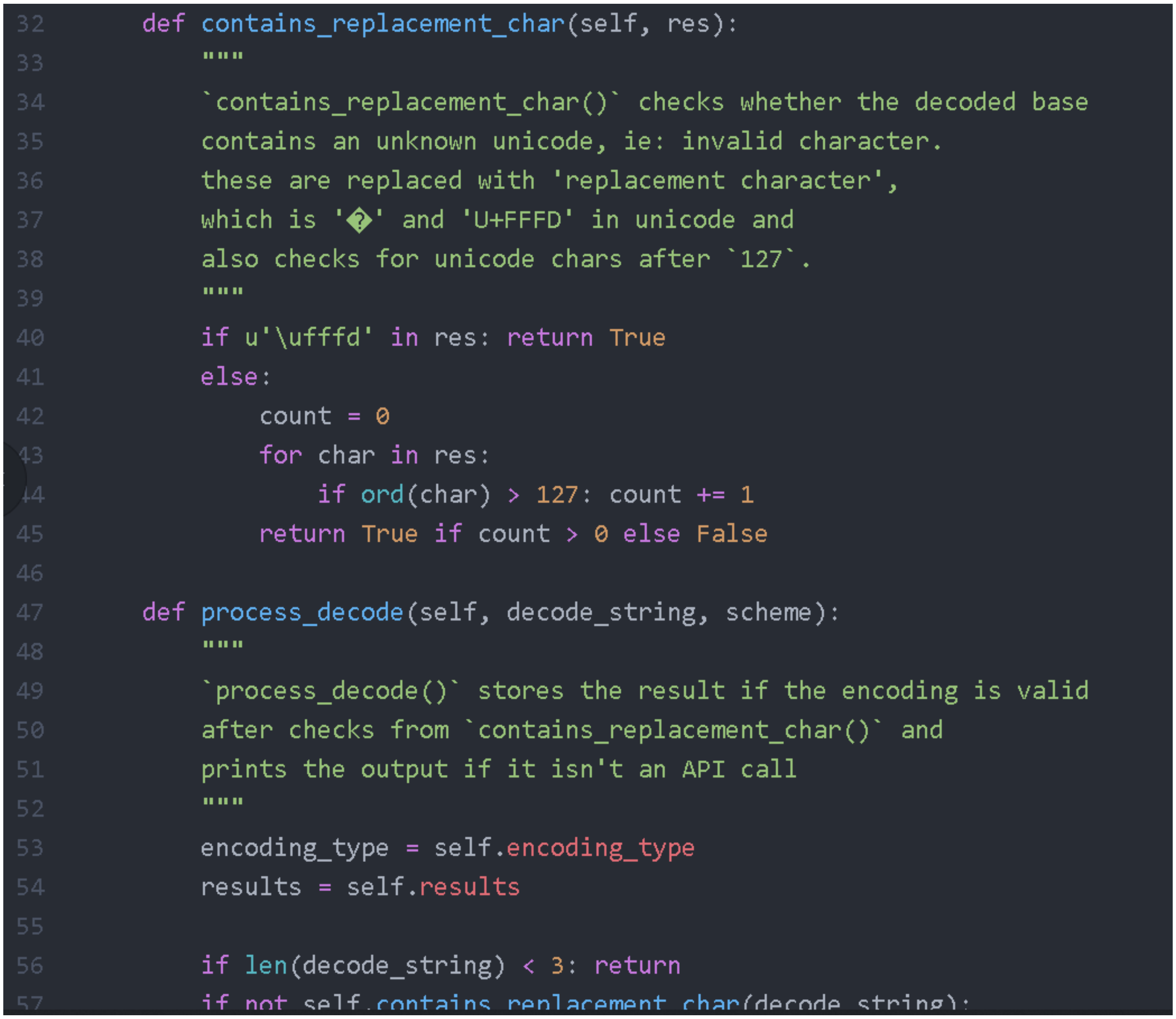


```
    if len(bstr) % 2 == 1:
        x = base92_ord(bstr[-1])
        bitstr += '{:06b}'.format(x)
        while 8 <= len(bitstr):
            resstr += chr(int(bitstr[0:8], 2))
            bitstr = bitstr[8:]
    return resstr

decode = base92_decode
b92decode = base92_decode
```

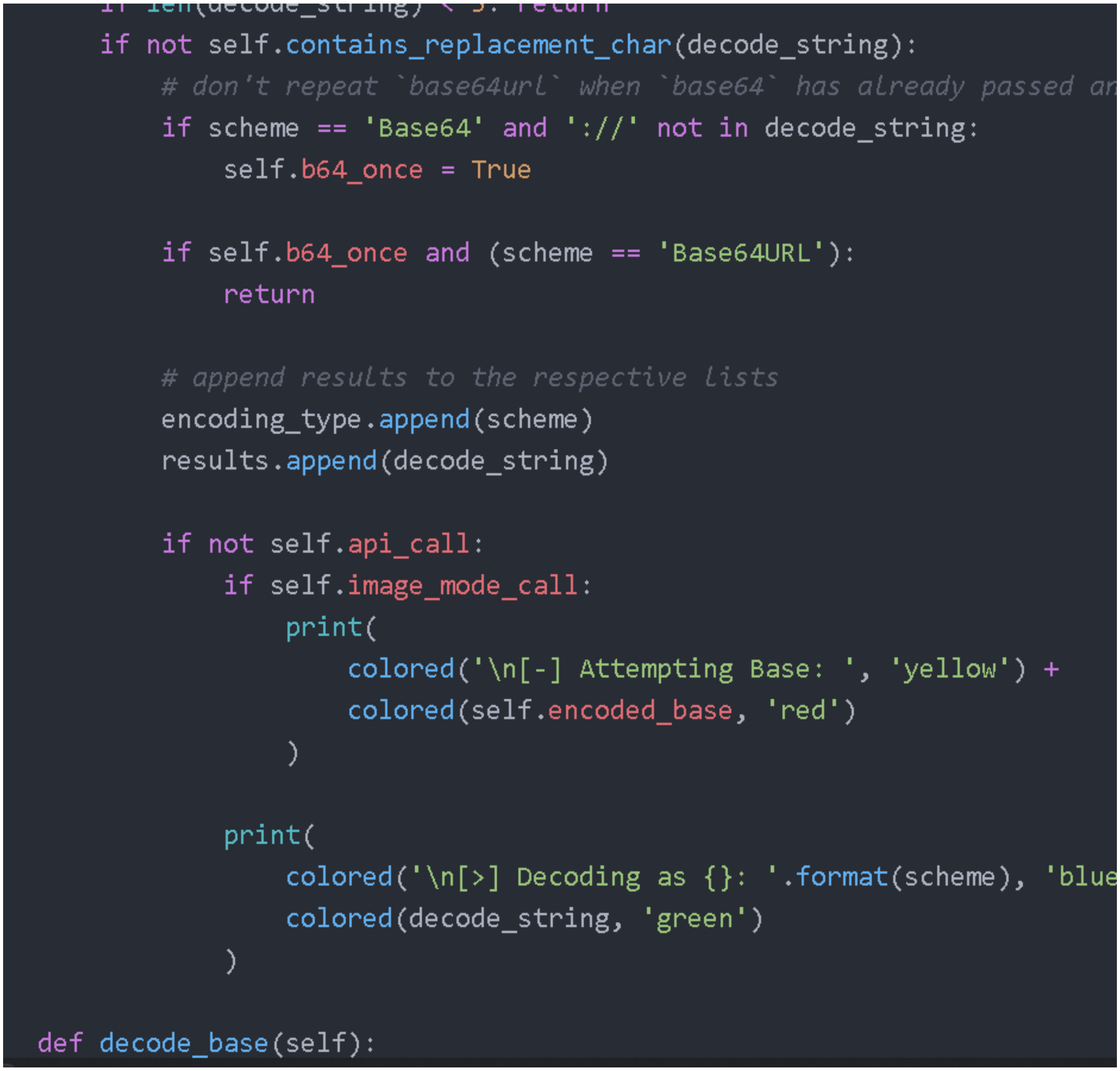
- **Base chain**



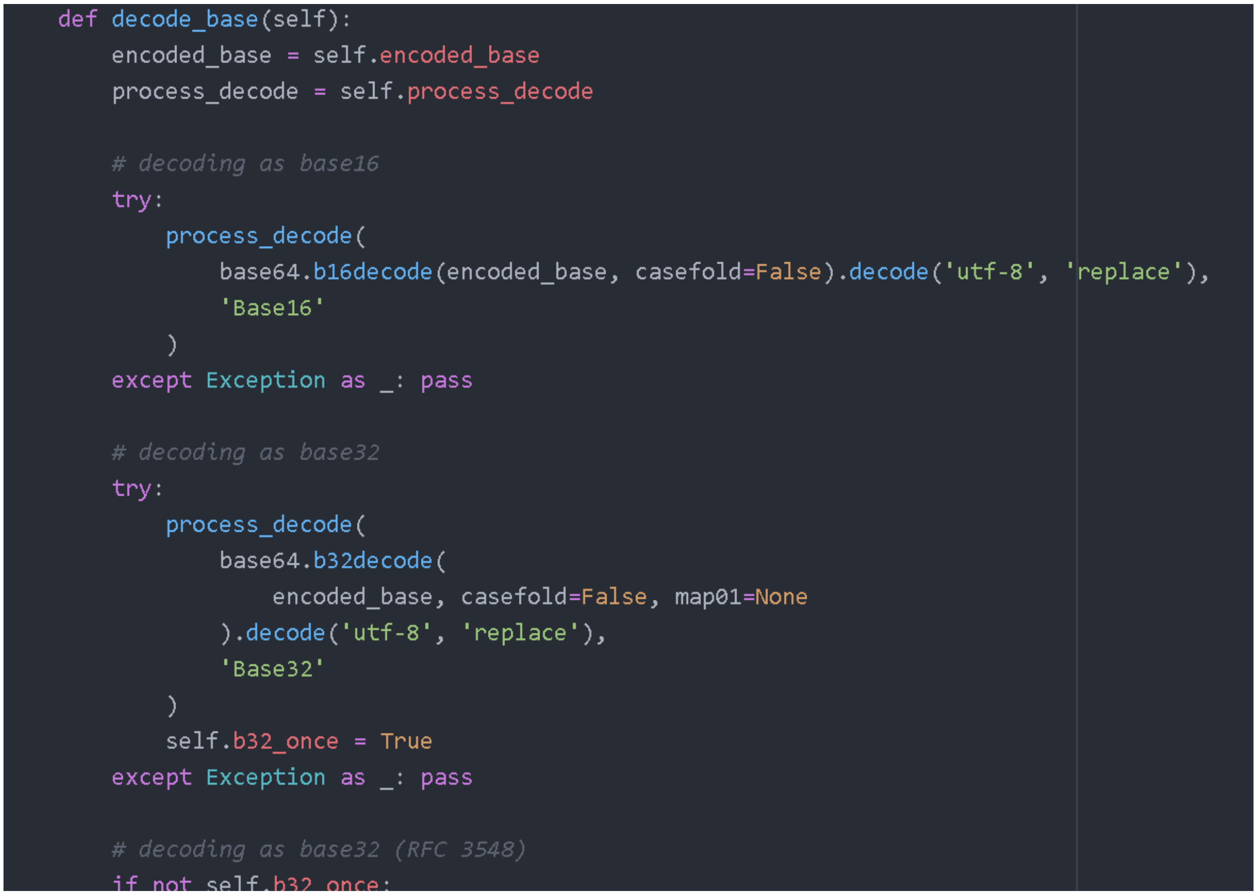
```python
import anybase32
import base36
import base58
import base62
import base64
import base91
import src.base92 as base92
import pybase100

from termcolor import colored

class DecodeBase:
    def __init__(self, encoded_base, api_call=False, image_mode=False):
        self.encoded_base = encoded_base
        self.b32_once = False
        self.b64_once = False
        self.b64_url = False
        self.encoding_type = []
        self.results = []

        # state conditions
        self.api_call = api_call
        self.image_mode_call = image_mode

    def decode(self):
        self.decode_base()
```



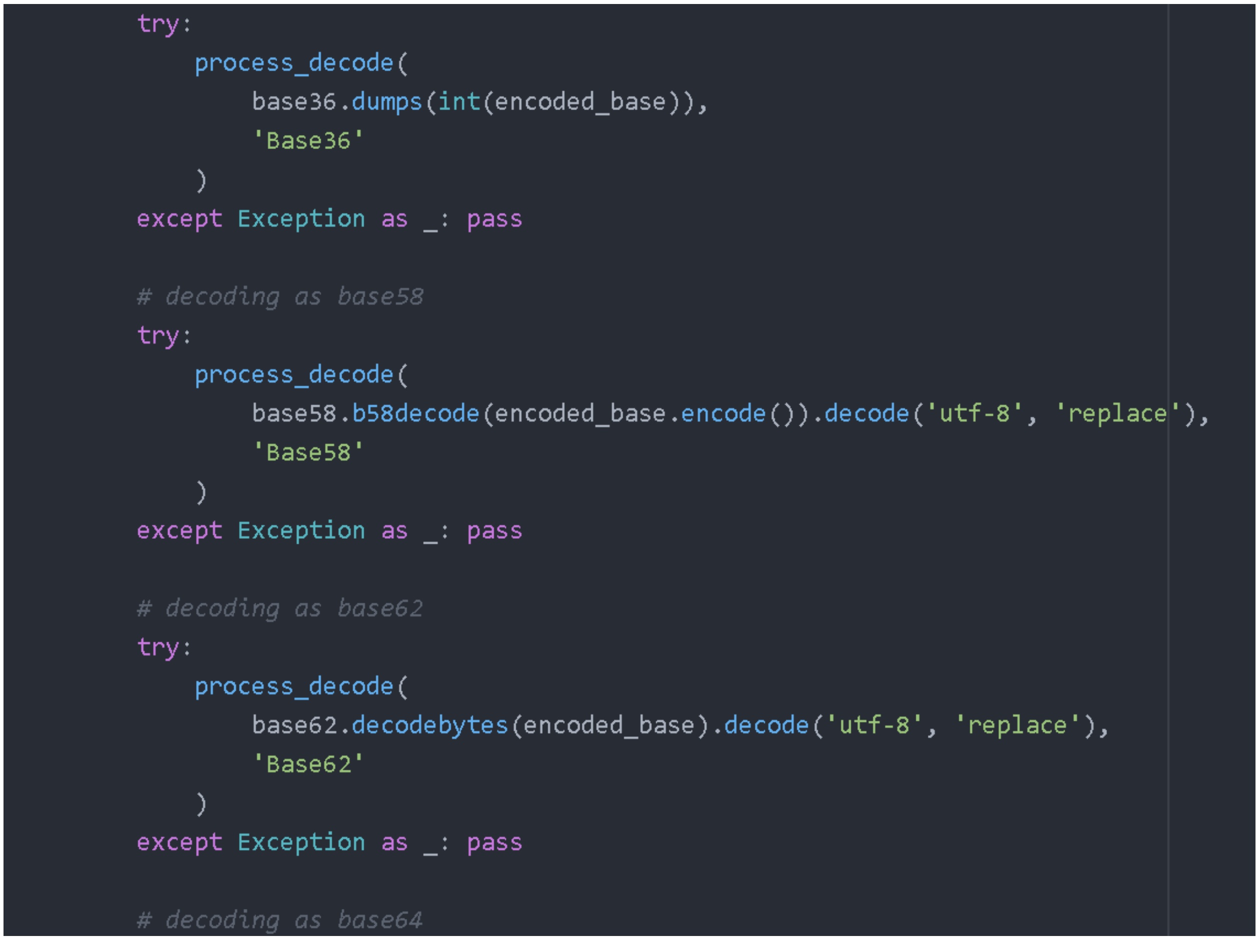
```python
    def contains_replacement_char(self, res):
        """
        `contains_replacement_char()` checks whether the decoded base
        contains an unknown unicode, ie: invalid character.
        these are replaced with 'replacement character',
        which is '�' and 'U+FFFD' in unicode and
        also checks for unicode chars after `127`.
        """
        if u'\ufffd' in res: return True
        else:
            count = 0
            for char in res:
                if ord(char) > 127: count += 1
            return True if count > 0 else False

    def process_decode(self, decode_string, scheme):
        """
        `process_decode()` stores the result if the encoding is valid
        after checks from `contains_replacement_char()` and
        prints the output if it isn't an API call
        """
        encoding_type = self.encoding_type
        results = self.results

        if len(decode_string) < 3: return
        if not self.contains_replacement_char(decode_string):
```



```
    if len(decode_string) < 3: return
    if not self.contains_replacement_char(decode_string):
        # don't repeat `base64url` when `base64` has already passed an
        if scheme == 'Base64' and '://' not in decode_string:
            self.b64_once = True

        if self.b64_once and (scheme == 'Base64URL'):
            return

        # append results to the respective lists
        encoding_type.append(scheme)
        results.append(decode_string)

        if not self.api_call:
            if self.image_mode_call:
                print(
                    colored('\n[-] Attempting Base: ', 'yellow') +
                    colored(self.encoded_base, 'red')
                )

            print(
                colored('\n[>] Decoding as {}: '.format(scheme), 'blue
                colored(decode_string, 'green')
            )

def decode_base(self):
```

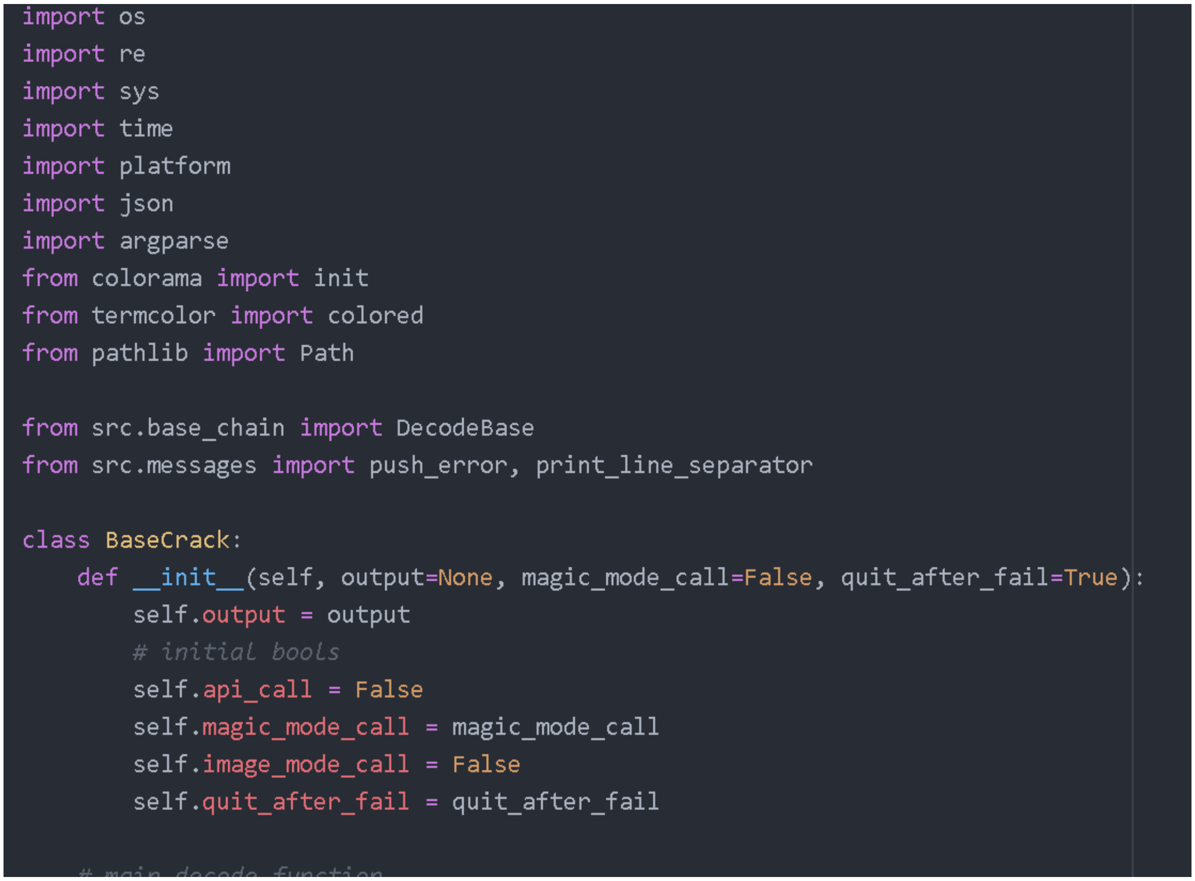


```
    def decode_base(self):
        encoded_base = self.encoded_base
        process_decode = self.process_decode

        # decoding as base16
        try:
            process_decode(
                base64.b16decode(encoded_base, casefold=False).decode('utf-8', 'replace'),
                'Base16'
            )
        except Exception as _: pass

        # decoding as base32
        try:
            process_decode(
                base64.b32decode(
                    encoded_base, casefold=False, map01=None
                ).decode('utf-8', 'replace'),
                'Base32'
            )
            self.b32_once = True
        except Exception as _: pass

        # decoding as base32 (RFC 3548)
        if not self.b32_once:
```

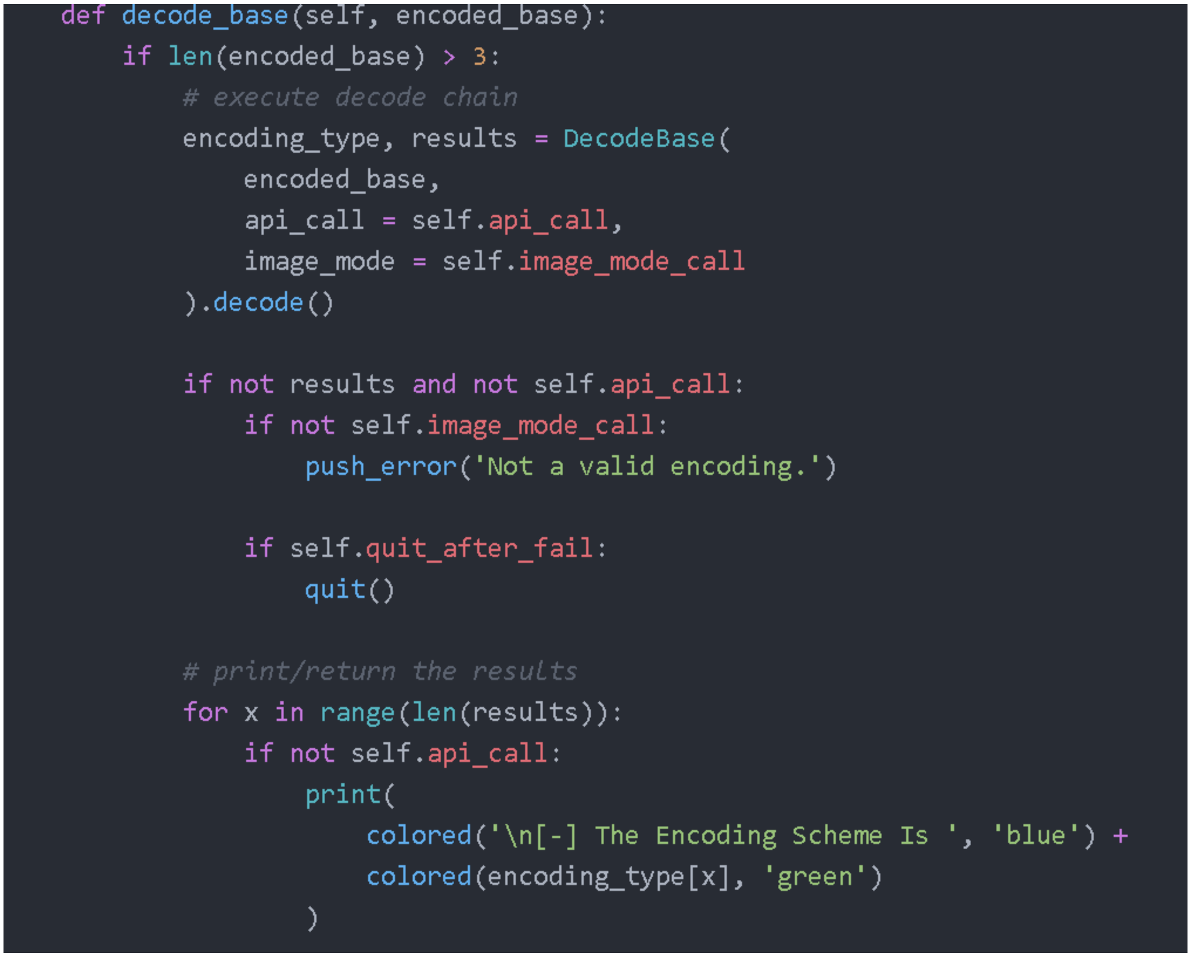


```
            try:
                process_decode(
                    base36.dumps(int(encoded_base)),
                    'Base36'
                )
            except Exception as _: pass

            # decoding as base58
            try:
                process_decode(
                    base58.b58decode(encoded_base.encode()).decode('utf-8', 'replace'),
                    'Base58'
                )
            except Exception as _: pass

            # decoding as base62
            try:
                process_decode(
                    base62.decodebytes(encoded_base).decode('utf-8', 'replace'),
                    'Base62'
                )
            except Exception as _: pass

            # decoding as base64
```

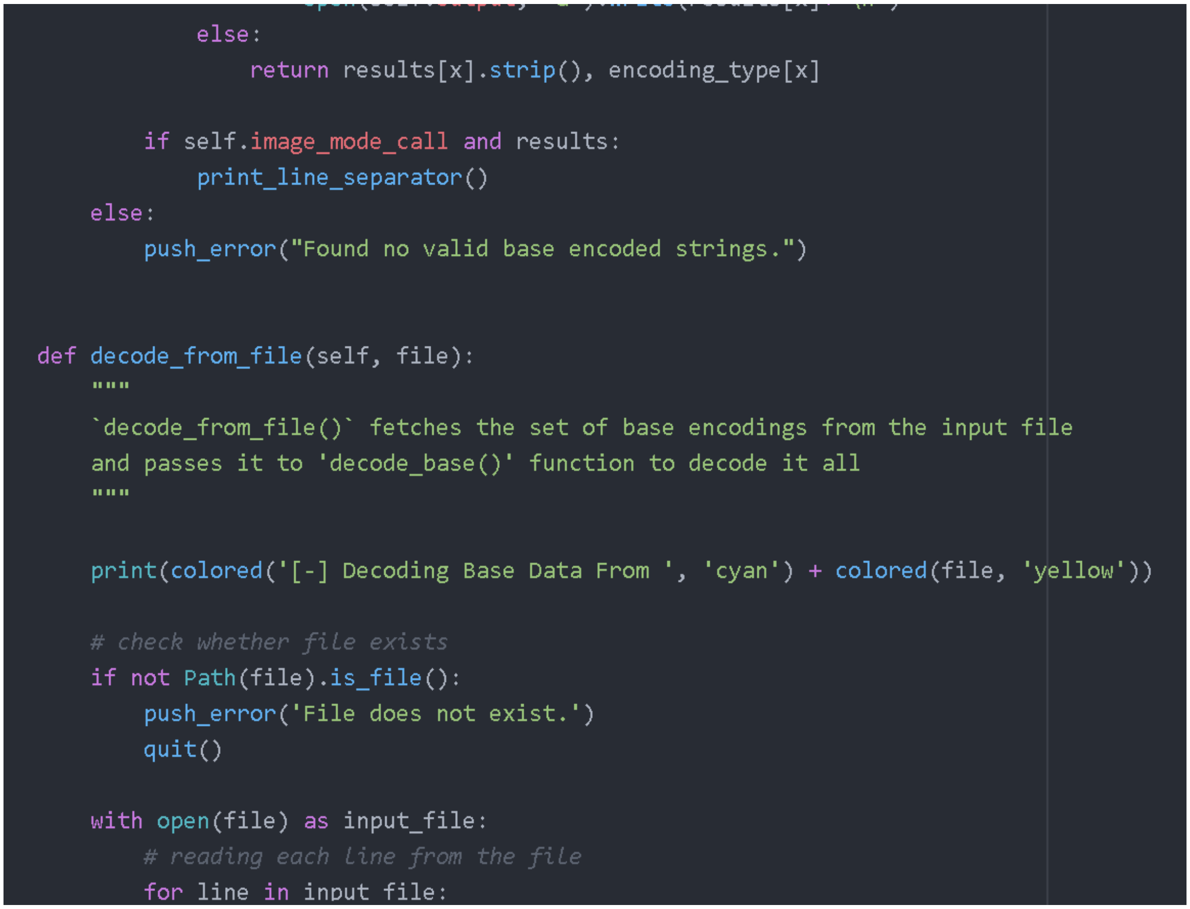
- **Decrypt_main**



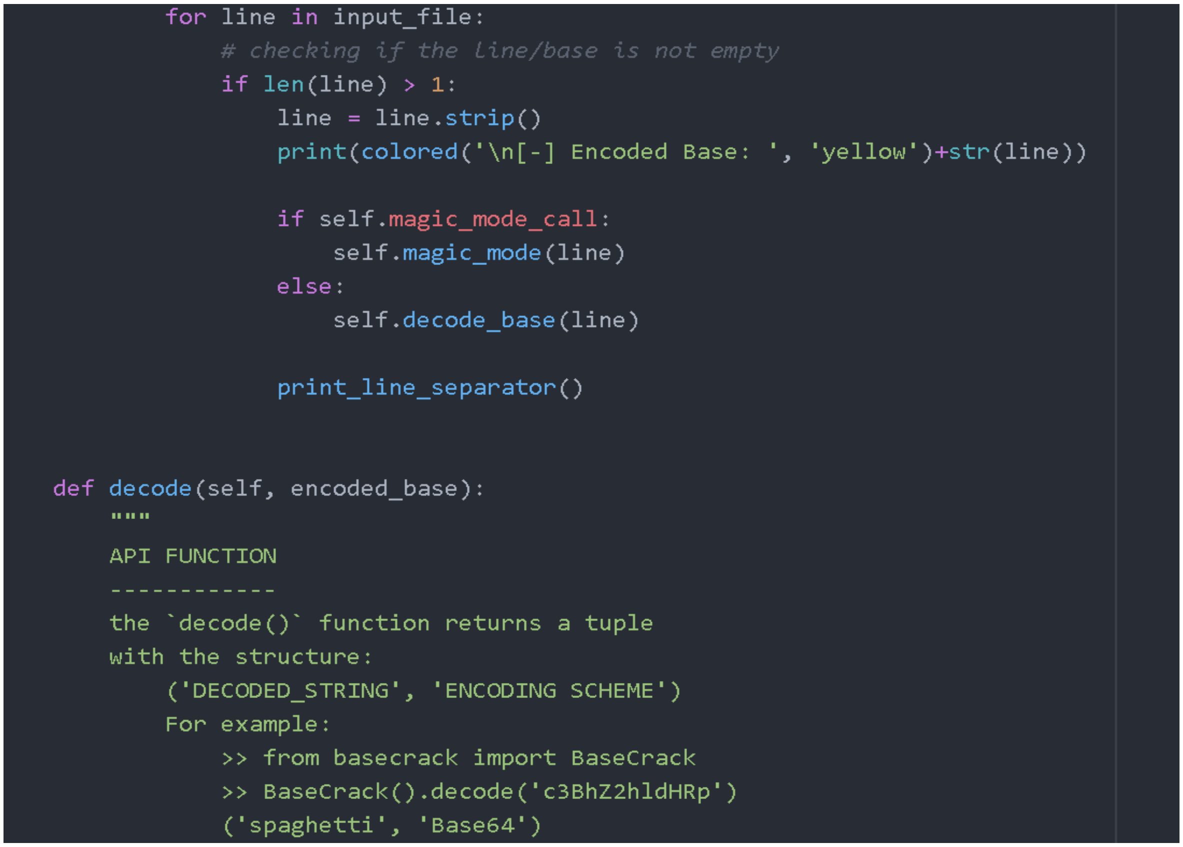
```
import os
import re
import sys
import time
import platform
import json
import argparse
from colorama import init
from termcolor import colored
from pathlib import Path

from src.base_chain import DecodeBase
from src.messages import push_error, print_line_separator

class BaseCrack:
    def __init__(self, output=None, magic_mode_call=False, quit_after_fail=True):
        self.output = output
        # initial bools
        self.api_call = False
        self.magic_mode_call = magic_mode_call
        self.image_mode_call = False
        self.quit_after_fail = quit_after_fail

    # main decode function
```



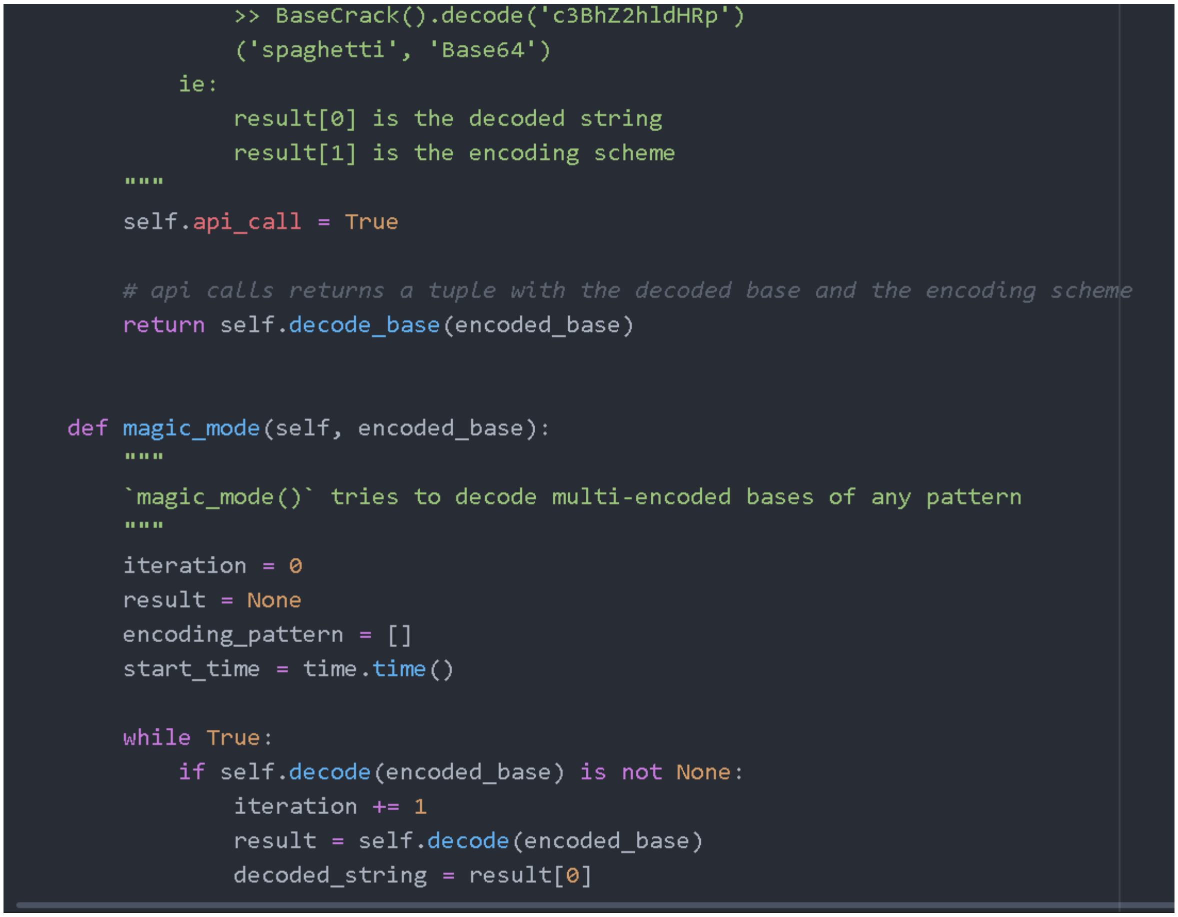
```
    def decode_base(self, encoded_base):
        if len(encoded_base) > 3:
            # execute decode chain
            encoding_type, results = DecodeBase(
                encoded_base,
                api_call = self.api_call,
                image_mode = self.image_mode_call
            ).decode()

            if not results and not self.api_call:
                if not self.image_mode_call:
                    push_error('Not a valid encoding.')

                if self.quit_after_fail:
                    quit()

            # print/return the results
            for x in range(len(results)):
                if not self.api_call:
                    print(
                        colored('\n[-] The Encoding Scheme Is ', 'blue') +
                        colored(encoding_type[x], 'green')
                    )
```



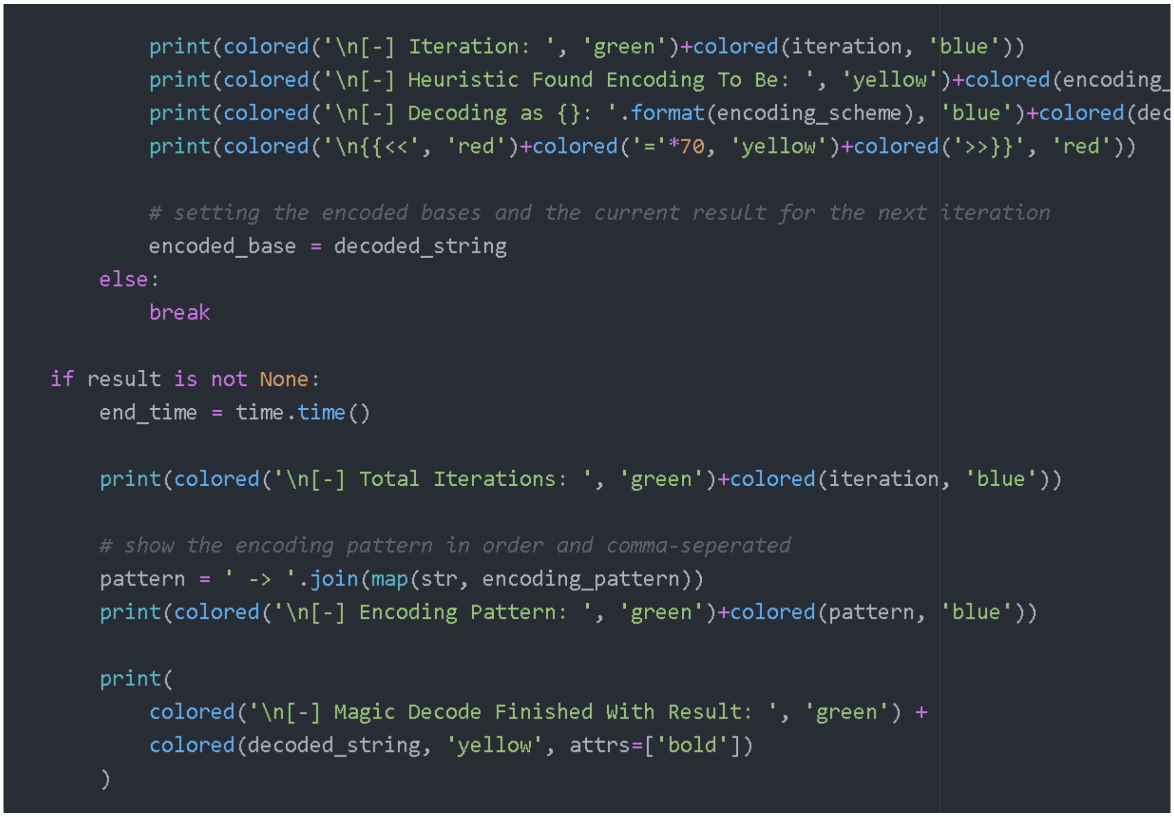
```
                        open(self.output, 'a').write(results[x]+'\n')
                else:
                    return results[x].strip(), encoding_type[x]

            if self.image_mode_call and results:
                print_line_separator()
        else:
            push_error("Found no valid base encoded strings.")


    def decode_from_file(self, file):
        """
        `decode_from_file()` fetches the set of base encodings from the input file
        and passes it to 'decode_base()' function to decode it all
        """

        print(colored('[-] Decoding Base Data From ', 'cyan') + colored(file, 'yellow'))

        # check whether file exists
        if not Path(file).is_file():
            push_error('File does not exist.')
            quit()

        with open(file) as input_file:
            # reading each line from the file
            for line in input_file:
```



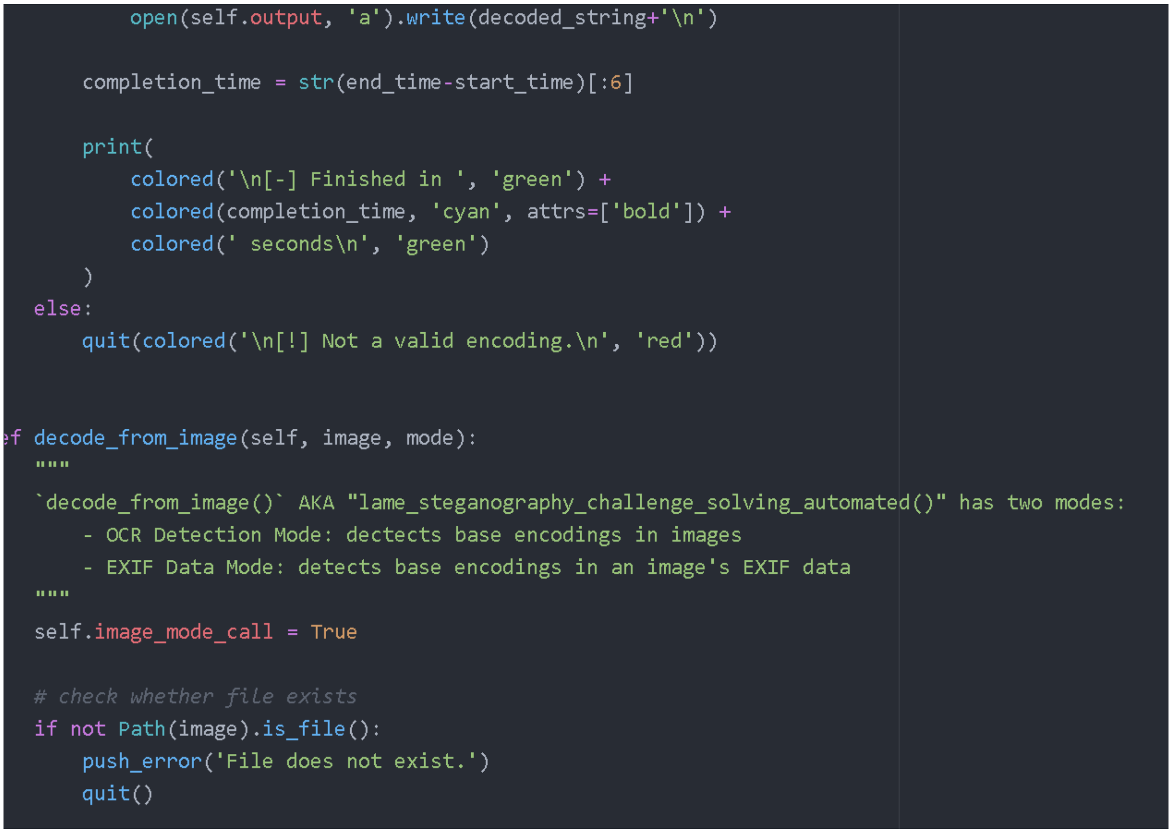
```
            for line in input_file:
                # checking if the line/base is not empty
                if len(line) > 1:
                    line = line.strip()
                    print(colored('\n[-] Encoded Base: ', 'yellow')+str(line))

                    if self.magic_mode_call:
                        self.magic_mode(line)
                    else:
                        self.decode_base(line)

                    print_line_separator()


    def decode(self, encoded_base):
        """
        API FUNCTION
        ------------
        the `decode()` function returns a tuple
        with the structure:
            ('DECODED_STRING', 'ENCODING SCHEME')
            For example:
                >> from basecrack import BaseCrack
                >> BaseCrack().decode('c3BhZ2hldHRp')
                ('spaghetti', 'Base64')
```



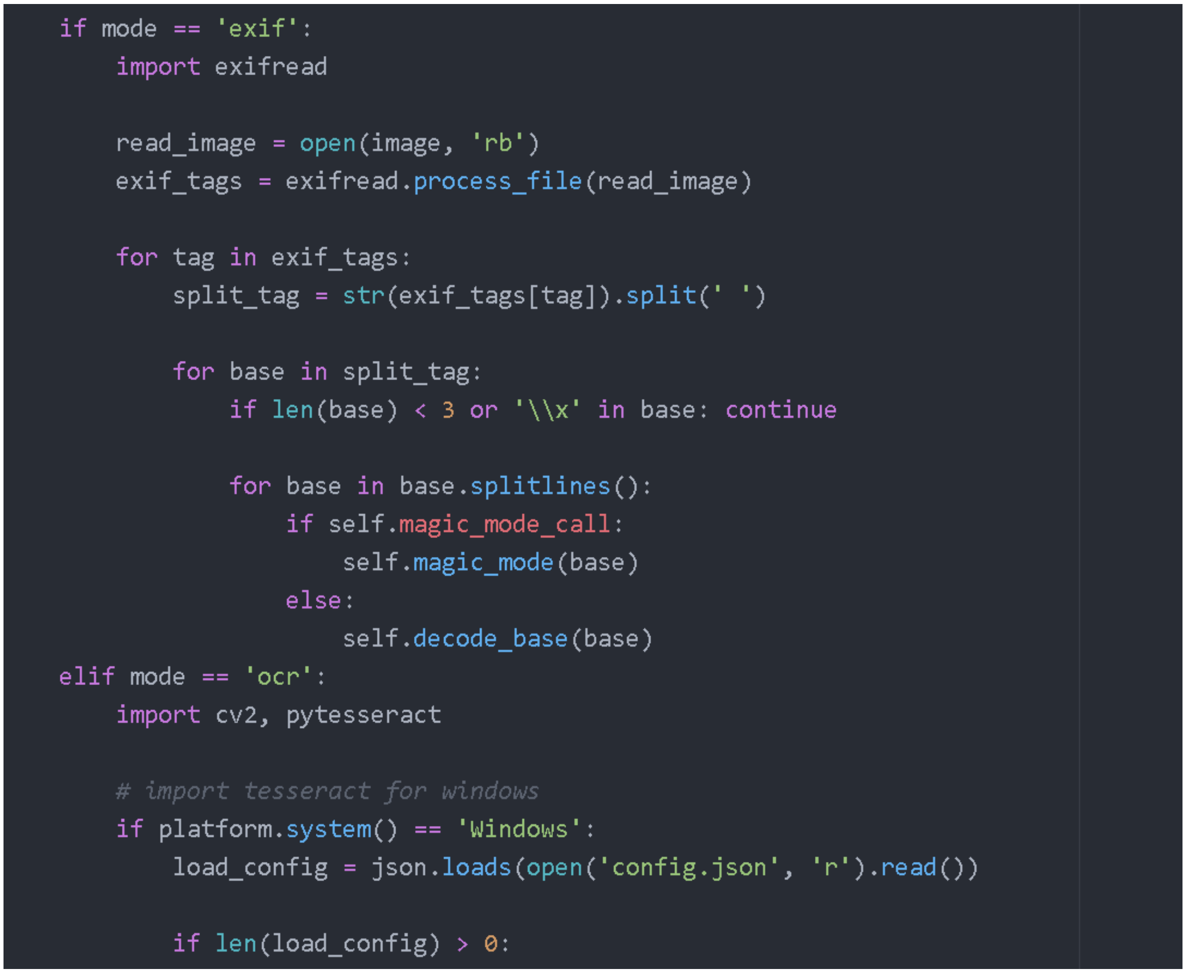
```
                >> BaseCrack().decode('c3BhZ2hldHRp')
                ('spaghetti', 'Base64')
            ie:
                result[0] is the decoded string
                result[1] is the encoding scheme
        """
        self.api_call = True

        # api calls returns a tuple with the decoded base and the encoding scheme
        return self.decode_base(encoded_base)


    def magic_mode(self, encoded_base):
        """
        `magic_mode()` tries to decode multi-encoded bases of any pattern
        """
        iteration = 0
        result = None
        encoding_pattern = []
        start_time = time.time()

        while True:
            if self.decode(encoded_base) is not None:
                iteration += 1
                result = self.decode(encoded_base)
                decoded_string = result[0]
```



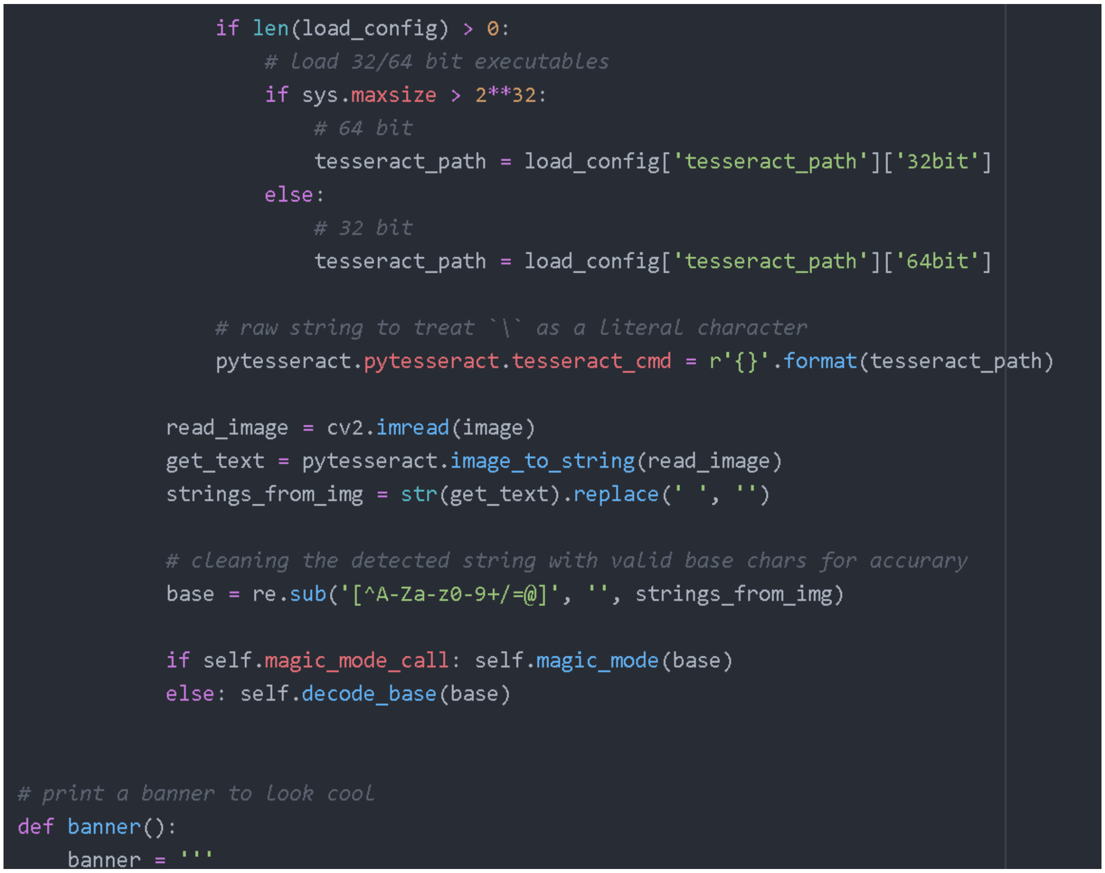
```
                print(colored('\n[-] Iteration: ', 'green')+colored(iteration, 'blue'))
                print(colored('\n[-] Heuristic Found Encoding To Be: ', 'yellow')+colored(encoding_
                print(colored('\n[-] Decoding as {}: '.format(encoding_scheme), 'blue')+colored(dec
                print(colored('\n{{<<', 'red')+colored('='*70, 'yellow')+colored('>>}}', 'red'))

                # setting the encoded bases and the current result for the next iteration
                encoded_base = decoded_string
            else:
                break

        if result is not None:
            end_time = time.time()

            print(colored('\n[-] Total Iterations: ', 'green')+colored(iteration, 'blue'))

            # show the encoding pattern in order and comma-seperated
            pattern = ' -> '.join(map(str, encoding_pattern))
            print(colored('\n[-] Encoding Pattern: ', 'green')+colored(pattern, 'blue'))

            print(
                colored('\n[-] Magic Decode Finished With Result: ', 'green') +
                colored(decoded_string, 'yellow', attrs=['bold'])
            )
```



```
            open(self.output, 'a').write(decoded_string+'\n')

        completion_time = str(end_time-start_time)[:6]

        print(
            colored('\n[-] Finished in ', 'green') +
            colored(completion_time, 'cyan', attrs=['bold']) +
            colored(' seconds\n', 'green')
        )
    else:
        quit(colored('\n[!] Not a valid encoding.\n', 'red'))


ef decode_from_image(self, image, mode):
    """
    `decode_from_image()` AKA "lame_steganography_challenge_solving_automated()" has two modes:
        - OCR Detection Mode: dectects base encodings in images
        - EXIF Data Mode: detects base encodings in an image's EXIF data
    """
    self.image_mode_call = True

    # check whether file exists
    if not Path(image).is_file():
        push_error('File does not exist.')
        quit()
```

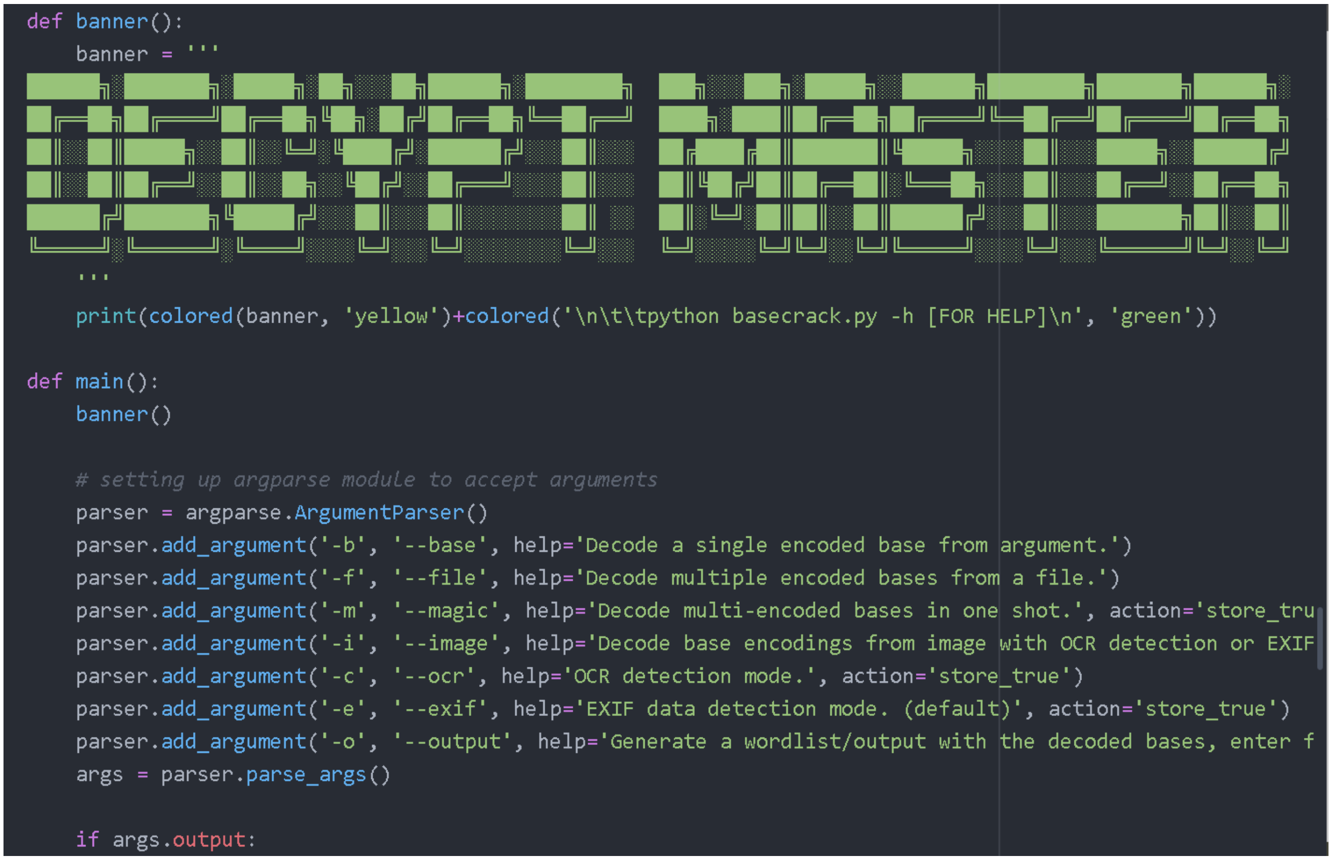


```
    if mode == 'exif':
        import exifread

        read_image = open(image, 'rb')
        exif_tags = exifread.process_file(read_image)

        for tag in exif_tags:
            split_tag = str(exif_tags[tag]).split(' ')

            for base in split_tag:
                if len(base) < 3 or '\\x' in base: continue

                for base in base.splitlines():
                    if self.magic_mode_call:
                        self.magic_mode(base)
                    else:
                        self.decode_base(base)
    elif mode == 'ocr':
        import cv2, pytesseract

        # import tesseract for windows
        if platform.system() == 'Windows':
            load_config = json.loads(open('config.json', 'r').read())

            if len(load_config) > 0:
```

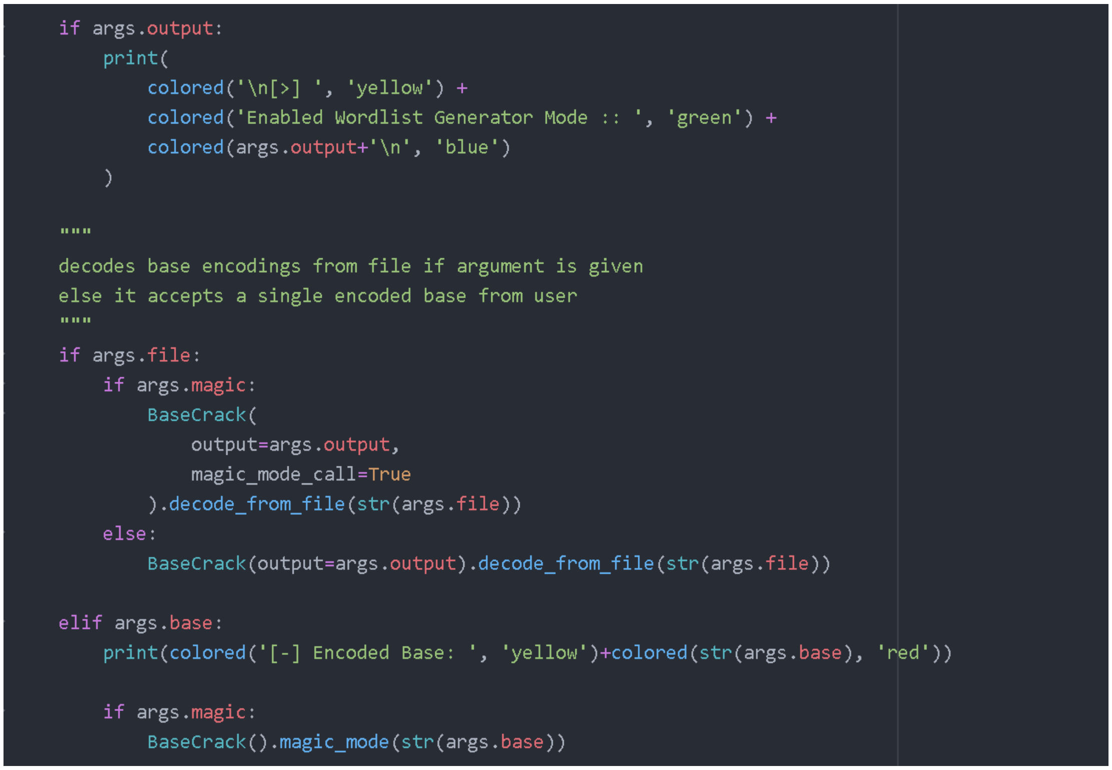


```
                if len(load_config) > 0:
                    # load 32/64 bit executables
                    if sys.maxsize > 2**32:
                        # 64 bit
                        tesseract_path = load_config['tesseract_path']['32bit']
                    else:
                        # 32 bit
                        tesseract_path = load_config['tesseract_path']['64bit']

                # raw string to treat `\` as a literal character
                pytesseract.pytesseract.tesseract_cmd = r'{}'.format(tesseract_path)

            read_image = cv2.imread(image)
            get_text = pytesseract.image_to_string(read_image)
            strings_from_img = str(get_text).replace(' ', '')

            # cleaning the detected string with valid base chars for accurary
            base = re.sub('[^A-Za-z0-9+/=@]', '', strings_from_img)

            if self.magic_mode_call: self.magic_mode(base)
            else: self.decode_base(base)


# print a banner to look cool
def banner():
    banner = '''
```

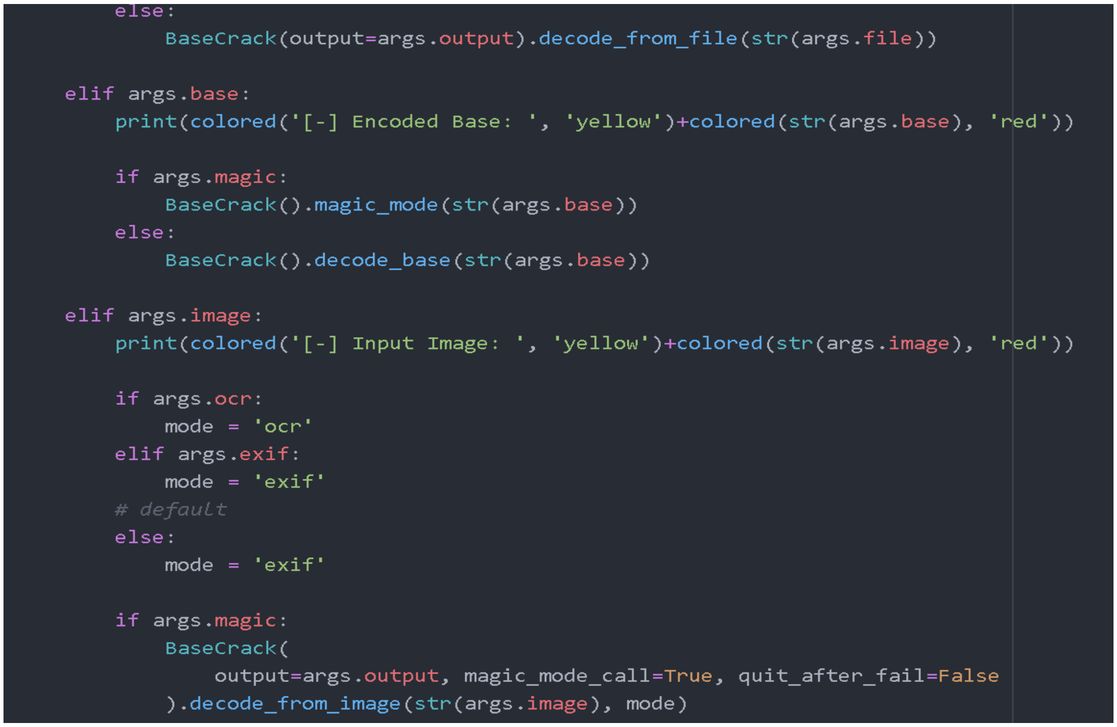
- **Banner Decrypt Master & Decode from image functions**



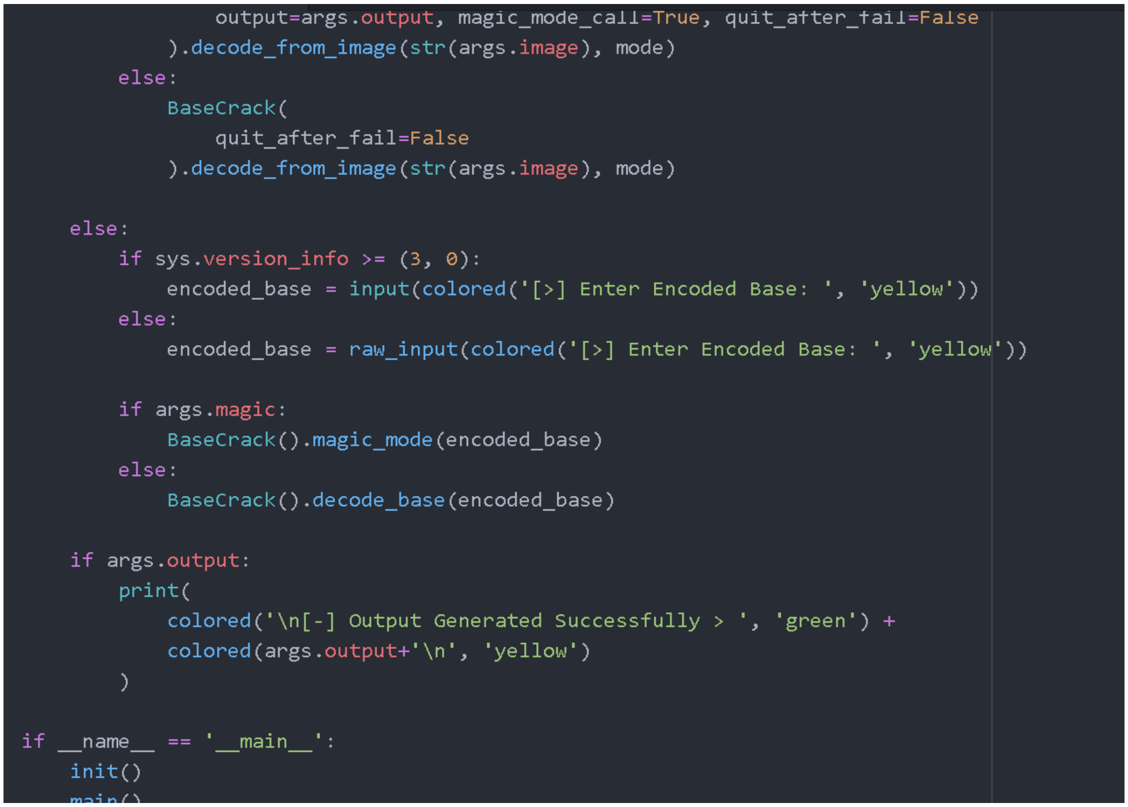
```
def banner():
    banner = '''
██████╗░███████╗░█████╗░██╗░░░██╗██████╗░████████╗  ███╗░░░███╗░█████╗░░██████╗████████╗███████╗██████╗░
██╔══██╗██╔════╝██╔══██╗╚██╗░██╔╝██╔══██╗╚══██╔══╝  ████╗░████║██╔══██╗██╔════╝╚══██╔══╝██╔════╝██╔══██╗
██║░░██║█████╗░░██║░░╚═╝░╚████╔╝░██████╔╝░░░██║░░░  ██╔████╔██║███████║╚█████╗░░░░██║░░░█████╗░░██████╔╝
██║░░██║██╔══╝░░██║░░██╗░░╚██╔╝░░██╔═══╝░░░░██║░░░  ██║╚██╔╝██║██╔══██║░╚═══██╗░░░██║░░░██╔══╝░░██╔══██╗
██████╔╝███████╗╚█████╔╝░░░██║░░░██║░░░░░░░░██║░░░  ██║░╚═╝░██║██║░░██║██████╔╝░░░██║░░░███████╗██║░░██║
╚═════╝░╚══════╝░╚════╝░░░░╚═╝░░░╚═╝░░░░░░░░╚═╝░░░  ╚═╝░░░░░╚═╝╚═╝░░╚═╝╚═════╝░░░░╚═╝░░░╚══════╝╚═╝░░╚═╝
    '''
    print(colored(banner, 'yellow')+colored('\n\t\tpython basecrack.py -h [FOR HELP]\n', 'green'))

def main():
    banner()

    # setting up argparse module to accept arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('-b', '--base', help='Decode a single encoded base from argument.')
    parser.add_argument('-f', '--file', help='Decode multiple encoded bases from a file.')
    parser.add_argument('-m', '--magic', help='Decode multi-encoded bases in one shot.', action='store_tru
    parser.add_argument('-i', '--image', help='Decode base encodings from image with OCR detection or EXIF
    parser.add_argument('-c', '--ocr', help='OCR detection mode.', action='store_true')
    parser.add_argument('-e', '--exif', help='EXIF data detection mode. (default)', action='store_true')
    parser.add_argument('-o', '--output', help='Generate a wordlist/output with the decoded bases, enter f
    args = parser.parse_args()

    if args.output:
```



```
    if args.output:
        print(
            colored('\n[>] ', 'yellow') +
            colored('Enabled Wordlist Generator Mode :: ', 'green') +
            colored(args.output+'\n', 'blue')
        )

    """
    decodes base encodings from file if argument is given
    else it accepts a single encoded base from user
    """
    if args.file:
        if args.magic:
            BaseCrack(
                output=args.output,
                magic_mode_call=True
            ).decode_from_file(str(args.file))
        else:
            BaseCrack(output=args.output).decode_from_file(str(args.file))

    elif args.base:
        print(colored('[-] Encoded Base: ', 'yellow')+colored(str(args.base), 'red'))

        if args.magic:
            BaseCrack().magic_mode(str(args.base))
```



```
        else:
            BaseCrack(output=args.output).decode_from_file(str(args.file))

    elif args.base:
        print(colored('[-] Encoded Base: ', 'yellow')+colored(str(args.base), 'red'))

        if args.magic:
            BaseCrack().magic_mode(str(args.base))
        else:
            BaseCrack().decode_base(str(args.base))

    elif args.image:
        print(colored('[-] Input Image: ', 'yellow')+colored(str(args.image), 'red'))

        if args.ocr:
            mode = 'ocr'
        elif args.exif:
            mode = 'exif'
        # default
        else:
            mode = 'exif'

        if args.magic:
            BaseCrack(
                output=args.output, magic_mode_call=True, quit_after_fail=False
            ).decode_from_image(str(args.image), mode)
```



```
                output=args.output, magic_mode_call=True, quit_after_fail=False
            ).decode_from_image(str(args.image), mode)
        else:
            BaseCrack(
                quit_after_fail=False
            ).decode_from_image(str(args.image), mode)

    else:
        if sys.version_info >= (3, 0):
            encoded_base = input(colored('[>] Enter Encoded Base: ', 'yellow'))
        else:
            encoded_base = raw_input(colored('[>] Enter Encoded Base: ', 'yellow'))

        if args.magic:
            BaseCrack().magic_mode(encoded_base)
        else:
            BaseCrack().decode_base(encoded_base)

    if args.output:
        print(
            colored('\n[-] Output Generated Successfully > ', 'green') +
            colored(args.output+'\n', 'yellow')
        )

if __name__ == '__main__':
    init()
    main()
```

6. References

1. **Wojciech Muła, Daniel Lemire, “Base64 encoding and decoding at almost the speed of a memory copy”.**

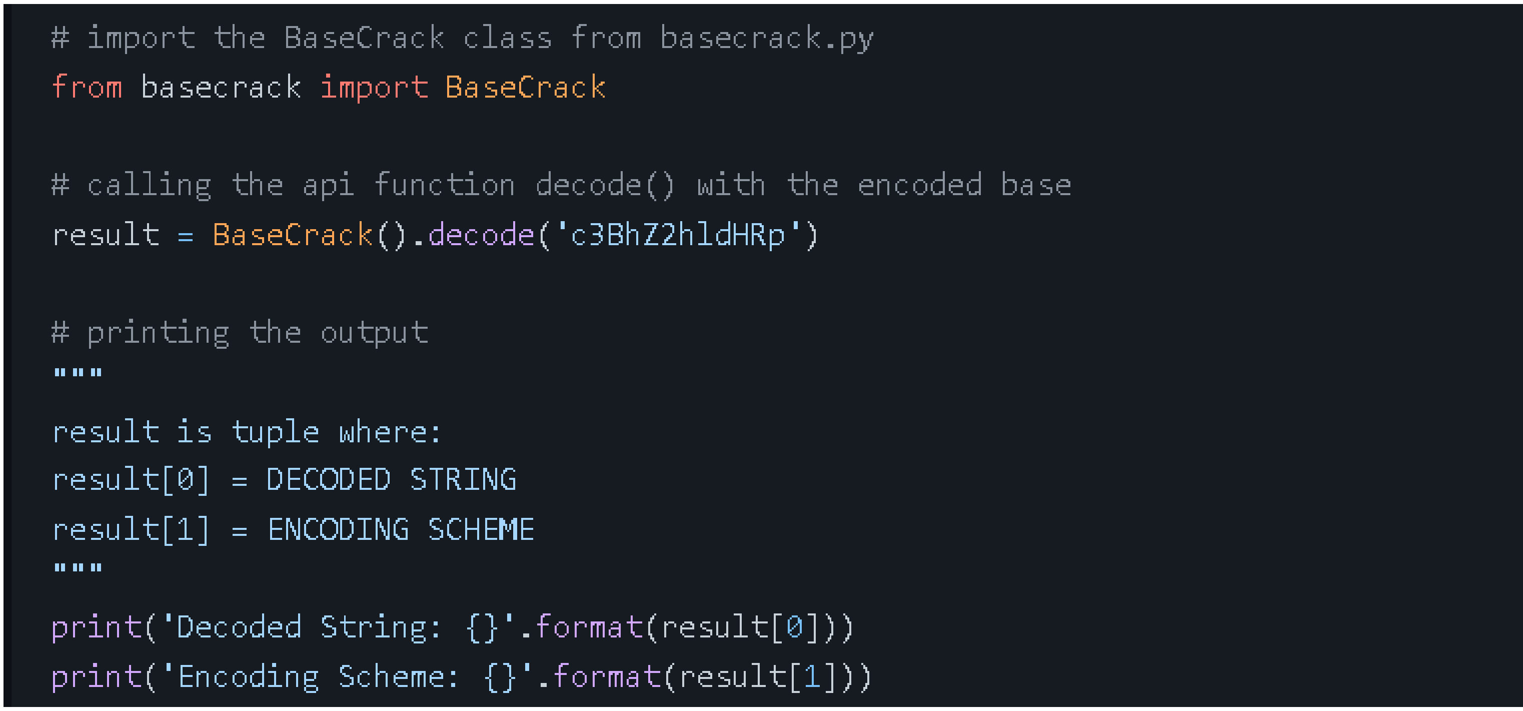
2. **Kenang Eko Prasetyo, Tito Waluyo Purboyo and Randy Erfa Saputra, “A Survey on Data Compression and Cryptographic Algorithms”.** *International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 23 (2017)*

3. **S. Josefsson, “The Base16, Base32, and Base64 Data Encodings”.** *The Internet Society (2003)*

4. **Mohammad A. Ahmad, Imad Fakhri Al Shaikhli, Hanady Mohammad Ahmad, ” Protection of the Texts Using Base64 and MD5”.** *Journal of Advanced Computer Science and Technology Research 2 (2012) 22-34*

Appendices

- As an A.P.I



```python
# import the BaseCrack class from basecrack.py
from basecrack import BaseCrack

# calling the api function decode() with the encoded base
result = BaseCrack().decode('c3BhZ2hldHRp')

# printing the output
"""
result is tuple where:
result[0] = DECODED STRING
result[1] = ENCODING SCHEME
"""
print('Decoded String: {}'.format(result[0]))
print('Encoding Scheme: {}'.format(result[1]))
```

7. Plagiarism Report

181430

ORIGINALITY REPORT

21 %	17%	2%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

4%

★ www.coursehero.com

Internet Source

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		