1. Browsers mainly track their user's activity to gain an inside look into what they are interested in and their habits. This is to provide a personalised experience to the user but can also be used to target them with specifically tailored advertising. Trackers will normally keep records of the websites you visit, what you purchase and for how long you were on these sites. They can also see which ads work for you/ones you click on which is in turn used to build an advertisement profile specific to you. Trackers can be blocked in a few different ways. On most modern browsers there will be an option in the privacy settings to send a "Do Not Track" request. This will send a request to the sites you visit that they don't track you and will in turn reduce the personalised experience you have. Some websites will also perform location requests which can be disabled by denying your browser access to location services. You can go a step further to increase your security online by downloading and installing extensions for your browser. uBlock origin is one of the most widely used extensions for privacy online. It is a free and open-source extension which can perform content filtering and blocking. Ghostery is an extension that extends the users ability to see what trackers are being applied to them. It monitors all the different web servers that you are connecting to and compares them to a list of known trackers. HTTPS Everywhere encrypts all communications so that it is more difficult for your data to be stolen. It provides, among other things, confirmation that the website you are connecting to is who it says it is.
   NordVPN is a virtual private network that keeps your IP address hidden to the public domain. It routes all your internet traffic though an encrypted connection to one of NordVPN's servers.

2.

```python
import string

def caesar_cipher(text, key, encrypt):
    if encrypt is True:
        key %= 26
        alphabet = string.ascii_uppercase
        # alphabet = string.ascii_lowercase
        keyed = alphabet[key:] + alphabet[:key]
        table = str.maketrans(alphabet, keyed)
        encrypted = text.translate(table)
        return encrypted

    elif encrypt is not True or key is None:
        for x in range(0, 26):
            key = 26 - x
            key %= 26
            print("-------------------------------------------------------------")
            print("Key value of : ", key)
            alphabet = string.ascii_uppercase
            # alphabet = string.ascii_lowercase
            keyed = alphabet[key:] + alphabet[:key]
            table = str.maketrans(alphabet, keyed)
            decrypted = text.translate(table)
            print(decrypted)


def vinegere_cipher(plaintext, key, encrypt):
    ciphertext = plaintext
    key = key
    def pad_the_key(plaintext, key):
        padded_key = ''
        i = 0
        for char in plaintext:
            if char.isalpha():
                padded_key += key[i % len(key)]
                i += 1
            else:
                padded_key += ' '
        return padded_key

    def encrypt_decrypt_char(plaintext_char, key_char):
        if plaintext_char.isalpha():
```

```
            first_alpha_letter = 'a'
            if plaintext_char.isupper():
                first_alpha_letter = 'A'

            old_char_position = ord(plaintext_char) - ord(first_alpha_letter)
            key_char_position = ord(key_char.lower()) - ord('a')

            if encrypt is True:
                new_char_position = (old_char_position + key_char_position) % 26
            else:
                new_char_position = (old_char_position - key_char_position + 26) % 26
            return chr(new_char_position + ord(first_alpha_letter))
        return plaintext_char

    def vinegere_encrypt(plaintext, key):
        ciphertext = ''
        padded_key = pad_the_key(plaintext, key)
        for plaintext_char, key_char in zip(plaintext, padded_key):
            ciphertext += encrypt_decrypt_char(plaintext_char, key_char)
        return ciphertext
```

```

    def vinegere_decrypt(ciphertext, key):
        plaintext = ''
        padded_key = pad_the_key(ciphertext, key)
        for ciphertext_char, key_char in zip(ciphertext, padded_key):
            plaintext += encrypt_decrypt_char(ciphertext_char, key_char)
        return plaintext
    if encrypt == True:
        print(vinegere_encrypt(ciphertext, key))
    else:
        print(vinegere_decrypt(ciphertext, key))


def main():...


main()

```

3. Key value of: 23
   ONE VARIATION TO THE STANDARD CAESAR CIPHER IS WHEN THE ALPHABET IS "KEYED" BY USING A WORD. IN THE TRADITIONAL VARIETY, ONE COULD WRITE THE ALPHABET ON TWO STRIPS AND JUST MATCH UP THE STRIPS AFTER SLIDING THE BOTTOM STRIP TO THE LEFT OR RIGHT. TO ENCODE, YOU WOULD FIND A LETTER IN THE TOP ROW AND SUBSTITUTE IT FOR THE LETTER IN THE BOTTOM ROW. FOR A KEYED VERSION, ONE WOULD NOT USE A STANDARD ALPHABET, BUT WOULD FIRST WRITE A WORD (OMITTING DUPLICATED LETTERS) AND THEN WRITE THE REMAINING LETTERS OF THE ALPHABET. FOR THE EXAMPLE BELOW, I USED A KEY OF "RUMKIN.COM" AND YOU WILL SEE THAT THE PERIOD IS REMOVED BECAUSE IT IS NOT A LETTER. YOU WILL ALSO NOTICE THE SECOND "M" IS NOT INCLUDED BECAUSE THERE WAS AN M ALREADY AND YOU CAN T HAVE DUPLICATES

4. Key value of: 9
   ONE VARIATION TO THE STANDARD CAESAR CIPHER IS WHEN THE ALPHABET IS "KEYED" BY USING A WORD. IN THE TRADITIONAL VARIETY, ONE COULD WRITE THE ALPHABET ON TWO STRIPS AND JUST MATCH UP THE STRIPS AFTER SLIDING THE BOTTOM STRIP TO THE LEFT OR RIGHT. TO ENCODE, YOU WOULD FIND A LETTER IN THE TOP ROW AND SUBSTITUTE IT FOR THE LETTER IN THE BOTTOM ROW. FOR A KEYED VERSION, ONE WOULD NOT USE A STANDARD ALPHABET, BUT WOULD FIRST WRITE A WORD (OMITTING DUPLICATED LETTERS) AND THEN WRITE THE REMAINING LETTERS OF THE ALPHABET. FOR THE EXAMPLE BELOW, I USED A KEY OF "RUMKIN.COM" AND YOU WILL SEE THAT THE PERIOD IS REMOVED BECAUSE IT IS NOT A LETTER. YOU WILL ALSO NOTICE THE SECOND "M" IS NOT INCLUDED BECAUSE THERE WAS AN M ALREADY AND YOU CAN T HAVE DUPLICATES.

5. Decrypted Plaintext:
   NIST IS ABOUT TO ANNOUNCE THE NEW HASH ALGORITHM THAT WILL BECOME SHA-3. THIS IS THE RESULT OF A SIX-YEAR COMPETITION, AND MY OWN SKEIN IS ONE OF THE FIVE REMAINING FINALISTS (OUT OF AN INITIAL 64). IT S PROBABLY TOO LATE FOR ME TO AFFECT THE FINAL DECISION, BUT I AM HOPING FOR "NO AWARD." IT S NOT THAT THE NEW HASH FUNCTIONS AREN T ANY GOOD, IT S THAT WE DON T REALLY NEED ONE. WHEN WE STARTED THIS PROCESS BACK IN 2006, IT LOOKED AS IF WE WOULD BE NEEDING A NEW HASH FUNCTION SOON. THE SHA FAMILY (WHICH IS REALLY PART OF THE MD4 AND MD5 FAMILY), WAS UNDER INCREASING PRESSURE FROM NEW TYPES OF CRYPTANALYSIS. WE DIDN T KNOW HOW LONG THE VARIOUS SHA-2 VARIANTS WOULD REMAIN SECURE. BUT IT S 2012, AND SHA-512 IS STILL LOOKING GOOD.

6.

```
alphabet = {
    "a": 0,
    "b": 1,
    "c": 2,
    "d": 3,
    "e": 4,
    "f": 5,
    "g": 6,
    "h": 7,
    "i": 8,
    "j": 9,
    "k": 10,
    "l": 11,
    "m": 12,
    "n": 13,
    "o": 14,
    "p": 15,
    "q": 16,
    "r": 17,
    "s": 18,
    "t": 19,
```

```
    "u": 20,
    "v": 21,
    "w": 22,
    "x": 23,
    "y": 24,
    "z": 25,
}
# for key, value in alphabet.items():
#     print(value)
key_matrix = [1, 0,
              10, 4]  # bake
plaintext = [2, 0,
             10, 4]  # cake

a = ((key_matrix[0] * plaintext[0]) + (key_matrix[1] * plaintext[2])) % 26
b = ((key_matrix[0] * plaintext[1]) + (key_matrix[1] * plaintext[3])) % 26
c = ((key_matrix[2] * plaintext[0]) + (key_matrix[3] * plaintext[2])) % 26
d = ((key_matrix[2] * plaintext[1]) + (key_matrix[3] * plaintext[3])) % 26


cipher_text = ""
for item in encrypted:
    for key, val in alphabet.items():
        if val == item:
            cipher_text += key

print("Plaintext: ", plaintext)
print("Encrypted text: ", encrypted)
print("Cipher text: " + cipher_text)

```