Assignment 3
Complementary materials

Dr. Weitao Xu

weitaoxu@cityu.edu.hk

Encrypt/decrypt a character

How to encrypt a character?

The following code encrypts a character `char` using a non-negative integer `key`.

[2]:
```
cc_n = 1114112


def cc_encrypt_character(char, key):
    """
    Return the encryption of a character by an integer key using Caesar cipher.

    Parameters
    ----------
    char: str
        a unicode (UTF-8) character to be encrypted.
    key int:
        secret key to encrypt char.
    """
    char_code = ord(char)
    shifted_char_code = (char_code + key) % cc_n
    encrypted_char = chr(shifted_char_code)
    return encrypted_char
```

For example, to encrypt the letter `'A'` using a secret key `5`:

[3]:
```
cc_encrypt_character("A", 5)
```

[3]:
```
'F'
```

ord() and chr()

- The ord() function returns the number representing the unicode code of a specified character.
- The chr() function returns the character that represents the specified unicode.

```
x=ord('a')
print(x)
y=chr(x)
print(y)
```

executed in 4ms, finished 08:34:18 2020

```
97
a
```

Encryption process

-

The character `'A'` is encrypted to the character `'F'` as follows:

1. `ord(char)` return the integer `65` that is the code point (integer representation) of the unicode of `'A'`.
2. `(char_code + key) % cc_n` cyclic shifts the code by the key `5`.
3. `chr(shifted_char_code)` converts the shifted code back to a character, which is `'F'`.

Encryption

`char`	...	**A**	B	C	D	E	F	...
`ord(char)`	...	**65**	66	67	68	69	70	...
`(ord(char) + key) % cc_n`	...	**70**	71	72	73	74	75	...
`(chr(ord(char) + key) % cc_n)`	...	**F**	G	H	I	J	K	...

How to decrypt a character?

Mathematically, we define the encryption and decryption of a character for Caesar cipher as

$$\begin{aligned} E(x,k) &:= x+k \mod n \quad (\text{encryption}) \\ D(x,k) &:= x-k \mod n \quad (\text{decryption}), \end{aligned} \tag{1}$$

where x is the character code in $\{0,\ldots,n\}$ and k is the secret key. `mod` operator above is the modulo operator. In Mathematics, it has a lower precedence than addition and multiplication and is typeset with an extra space accordingly.

The encryption and decryption satisfies the recoverability condition

$$D(E(x,k),k) = x \tag{2}$$

so two people with a common secret key can encrypt and decrypt a character, but others not knowing the key cannot. This is a defining property of a *symmetric cipher*.

The following code decrypts a character using a key.

```
def cc_decrypt_character(char, key):
    """
    Return the decryption of a character by the key using Caesar cipher.

    Parameters
    ==========
    char: str
        a unicode (UTF-8) character to be decrypted.
    key: int
        secret key to decrypt char.
    """
    char_code = ord(char)
    shifted_char_code = (char_code - key) % cc_n
    decrypted_char = chr(shifted_char_code)
    return decrypted_char
```

For instance, to decrypt the letter `'F'` by the secret key `5`:

```
cc_decrypt_character("F", 5)
```

```
'A'
```

The character `'F'` is decrypted back to `'A'` because `(char_code - key) % cc_n` reverse cyclic shifts the code by the key `5`.

Encryption								**Decryption**
`char`	…	A	B	C	D	E	F	…	`(chr(ord(char) - key) % cc_n)`
`ord(char)`	…	**65**	66	67	68	69	70	…	`(ord(char) - key) % cc_n`
`(ord(char) + key) % cc_n`	…	**70**	71	72	73	74	75	…	`ord(char)`
`(chr(ord(char) + key) % cc_n)`	…	**F**	G	H	I	J	K	…	`char`

Think: Why did we set `cc_n = 1114112`? Explain whether the recoverability property may fail if we set `cc_n` to a bigger number or remove `% cc_n` for both `cc_encrypt_character` and `cc_decrypt_character`.

Solution: `cc_n` is set to be the number of unicode characters. `ord` returns a code point between `0` and `cc_n-1`, so *the modulo operator ensures the shifted character code* `shifted_char_code` *remains a valid character code.* If we set `cc_n` to a bigger number or remove the modular operation, the code can fail because `shifted_char_code` may not be a valid code. E.g., `chr(1114112)` causes a ValueError.

Encrypt a plaintext and decrypt a ciphertext

Of course, it is more interesting to encrypt a string instead of a character. The following code implements this in one line.

[ ]:
```
def cc_encrypt(plaintext, key):
    """
    Return the ciphertext of a plaintext by the key using the Caesar cipher.

    Parameters
    ----------
    plaintext: str
        A unicode (UTF-8) message to be encrypted.
    public_key: int
        Public key to encrypt plaintext.
    """
    return "".join([chr((ord(char) + key) % cc_n) for char in plaintext])
```

The above function encrypts a message, referred to as the *plaintext*, by replacing each character with its encryption.
This is referred to as a *substitution cipher*.

Exercise 1 (2 marks)

Define a function `cc_decrypt` that

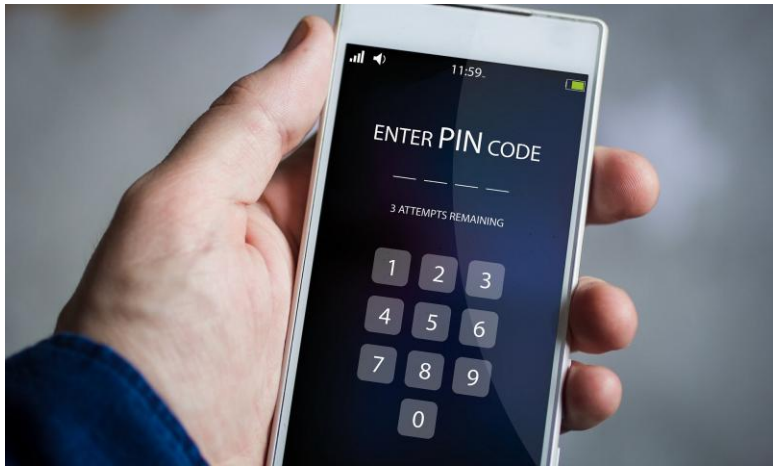
- takes a string `ciphertext` and an integer `key`, and
- returns the plaintext that encrypts to `ciphertext` by the key using Caesar cipher.

[9]:
```
def cc_decrypt(ciphertext, key):
    """
    Return the plaintext that encrypts to ciphertext by the key using Caesar cipher.

    Parameters
    ==========
    ciphertext: str
        message to be decrypted.
    key: int
        secret key to decrypt the ciphertext.
    """
    # YOUR CODE HERE
    pass
```

Next, we learn how to break a cipher

- In cryptography, a brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly.

0000
1111
…
1234
2345
3456
6789
…..
9999



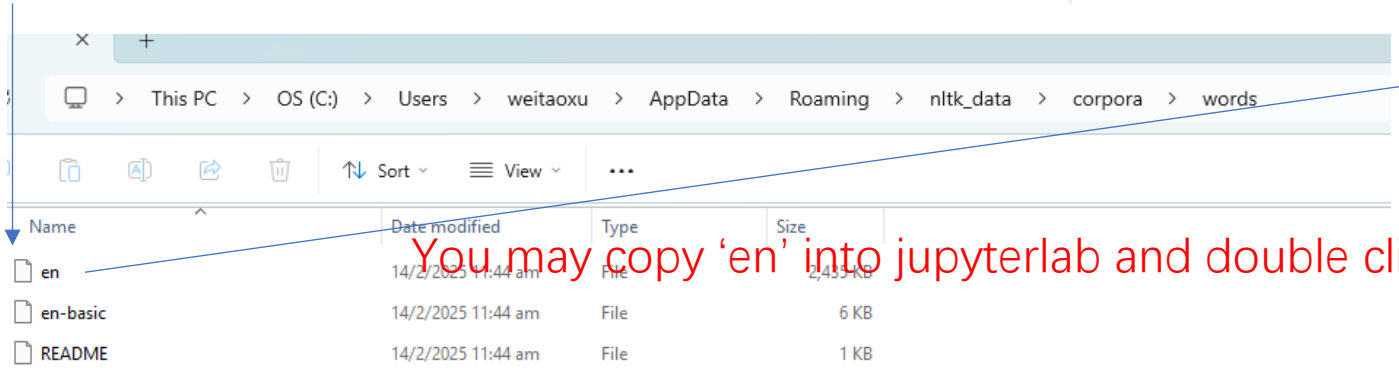
Run these two cells to download a package to your local computer

```
[11]: #please run this cell to install nltk, this process may take a while
!pip install nltk
```

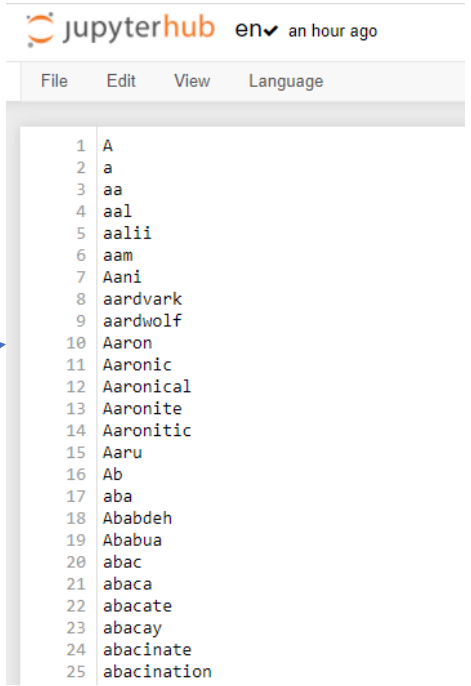
```
Collecting nltk
  Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
     ---------------------------------------- 1.5/1.5 MB 13.6 MB/s eta 0:00:00
Collecting click
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
     ---------------------------------------- 98.2/98.2 kB 5.5 MB/s eta 0:00:00
Collecting joblib
  Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
     ---------------------------------------- 301.8/301.8 kB 19.4 MB/s eta 0:00:00
Collecting regex>=2021.8.3
  Downloading regex-2024.11.6-cp311-cp311-win_amd64.whl (274 kB)
     ---------------------------------------- 274.1/274.1 kB 16.5 MB/s eta 0:00:00
Collecting tqdm
  Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
     ---------------------------------------- 78.5/78.5 kB ? eta 0:00:00
Requirement already satisfied: colorama in c:\users\weitaoxu\appdata\roaming\python\python311\site-packages (from click->nltk) (0.
4.6)
Installing collected packages: tqdm, regex, joblib, click, nltk
Successfully installed click-8.1.8 joblib-1.4.2 nltk-3.9.1 regex-2024.11.6 tqdm-4.67.1

[notice] A new release of pip available: 22.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
[12]: #this cell will download a package words to your local computer
import nltk

nltk.download("words")
from nltk.corpus import words
```

```
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\weitaoxu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\words.zip.
```



You may copy ‘en’ into jupyterlab and double click to open it



Now, let's define a funtion `tokenizer` that

- takes a string `text` as an argument, and
- returns a `list` of tokens obtained by
    1. splitting `text` into a list using `split()`;
    2. removing leading/trailing punctuations in `string.punctuation` using the `strip` method; and
    3. converting all items of the list to lowercase using `lower()`.

20]:
```python
import string


def tokenizer(text):
    """Returns the list of tokens of the text such that
    1) each token has no leading or training spaces/punctuations, and
    2) all letters in each tokens are in lowercase."""
    return [token.strip(string.punctuation).lower() for token in text.split()]
```

21]:
```python
# tests
assert tokenizer("Hello, World!") == ["hello", "world"]
assert get_score("Hello, World!") >= 0.99999
assert tokenizer("Do you know Jean-Pierre?") == ["do", "you", "know", "jean-pierre"]
assert get_score("Do you know Jean-Pierre?") >= 0.99999
```

The above code uses string.punctuation

- In Python, string.punctuation will give the all sets of punctuation.
- Use it directly, it's not a function, i.e., string.punctuation is correct but string.punctuation() is wrong
- More information here https://www.geeksforgeeks.org/string-punctuation-in-python/

In [5]:
```
import string
all_punctuation =  string.punctuation
print(all_punctuation)

s1="!hello world!"
#this is how to remove all leading/trailing punctuations
s2=s1.strip(string.punctuation)
print(s1)
print(s2)
```

executed in 4ms, finished 13:34:13 2021-04-08

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
!hello world!
hello world
```

Launch a brute-force attack

Exercise 2 (3 marks)

Define the function `cc_attack` that

- takes as arguments
  - a string `ciphertext`,
  - a floating point number `threshold` in the interval $(0, 1)$ with a default value of 0.6, and
- returns a generator that
  - generates one-by-one in ascending order guesses of the key that
  - decrypt `ciphertext` to texts with scores at least the `threshold`.

Note: returns a generator, so in the function you need to use yield, not return

```
4]: def cc_attack(ciphertext, threshold=0.6):
        """Returns a generator that generates the next guess of the key that
        decrypts the ciphertext to a text with get_score(text) at least the threshold.
        """
        # YOUR CODE HERE
        pass
```

If your code is correct, this
Is what you get after running the tests

```
[19]: # tests
ciphertext = cc_encrypt("Hello, World!", 12345)
key_generator = cc_attack(ciphertext)
key_guess = next(key_generator)
assert key_guess == 12345
text = cc_decrypt(ciphertext, key_guess)
print(
    "guess of the key: {}\nscore: {}\ntext :{}".format(key_guess, get_score(text), text)
)
```

```
guess of the key: 12345
score: 1.0
text :Hello, World!
```

This is a challenging question, read the materials to help you (it's a bit difficult ☺)

Columnar Transposition Cipher

Another symmetric key cipher is *columnar transposition cipher*. A transposition cipher encrypts a text by permuting instead of substituting characters.

Exercise 3 (5 marks)
Study and implement the irregular case of the *columnar transposition cipher* as described in Wikipedia page. Define the functions

- `ct_encrypt(plaintext, key)` for encryption, and
- `ct_decrypt(ciphertext, key)` for decryption.

You can assume the plaintext is in uppercase and has no spaces/punctuations.

Hints: See the test cases for an example of `plaintext`, `key`, and the corresponding `ciphertext`. You can but are not required to follow the solution template below:

```
def argsort(seq):
    '''A helper function that returns the tuple of indices that would sort the
    sequence seq.'''
    return tuple(x[0] for x in sorted(enumerate(seq), key=lambda x: x[1]))


def ct_idx(length, key):
    '''A helper function that returns the tuple of indices that would permute
    the letters of a message according to the key using the irregular case of
    columnar transposition cipher.'''
    seq = tuple(range(length))
    return [i for j in argsort(key) for i in _______________]


def ct_encrypt(plaintext, key):
    """
    Return the ciphertext of a plaintext by the key using the irregular case
```

Let's analyze the code step-by-step

```
def argsort(seq):
    '''A helper function that returns the tuple of indices that would sort the
    sequence seq.'''
    return tuple(x[0] for x in sorted(enumerate(seq), key=lambda x: x[1]))

key = 'ZEBRAS'
plaintext = 'WEAREDISCOVEREDFLEEATONCE'
ciphertext = 'EVLNACDTESEAROFODEECWIREE'
print(argsort(key))
```

```
(4, 2, 1, 3, 5, 0)
```

Running the above code (do not create this cell in the assignment, you may copy the assignment and create the above code in the copied version),
you'll get:

(4, 2, 1, 3, 5, 0) → What does it mean?

key: Z E B R A S
index: 0 1 2 3 4 5

after sort: A B E R S Z

so the index of the sorted sequence is: 4 2 1 3 5 0

Hints: See the test cases for an example of `plaintext`, `key`, and the corresponding `ciphertext`. You can but are not required to follow the solution template below:

```
def argsort(seq):
    '''A helper function that returns the tuple of indices that would sort the
    sequence seq.'''
    return tuple(x[0] for x in sorted(enumerate(seq), key=lambda x: x[1]))


def ct_idx(length, key):
    '''A helper function that returns the tuple of indices that would permute
    the letters of a message according to the key using the irregular case of
    columnar transposition cipher.'''
    seq = tuple(range(length))
    return [i for j in argsort(key) for i in _______________]


def ct_encrypt(plaintext, key):
    """
    Return the ciphertext of a plaintext by the key using the irregular case
    of columnar transposition cipher.

    Parameters
    ==========
    plaintext: str
        a message in uppercase without punctuations/spaces.
    key: str
        secret key to encrypt plaintext.
    """
    return ''.join([plaintext[i] for i in ct_idx(len(plaintext), key)])


def ct_decrypt(ciphertext, key):
    """
    Return the plaintext of the ciphertext by the key using the irregular case
    of columnar transposition cipher.

    Parameters
    ==========
    ciphertext: str
        a string in uppercase without punctuations/spaces.
    key: str
        secret key to decrypt ciphertext.
    """
    return ____________________________________________________________________
```

Complete this function

Complete this function

[3]:
```
def ct_idx(length, key):
    '''A helper function that returns the tuple of indices that would permute
    the letters of a message according to the key using the irregular case of
    columnar transposition cipher.'''
    seq = tuple(range(length))
    return [i for j in argsort(key) for i in ...]

key = 'ZEBRAS'
plaintext = 'WEAREDISCOVEREDFLEEATONCE'
ciphertext = 'EVLNACDTESEAROFODEECWIREE'
print(ct_idx(len(plaintext), key))
```

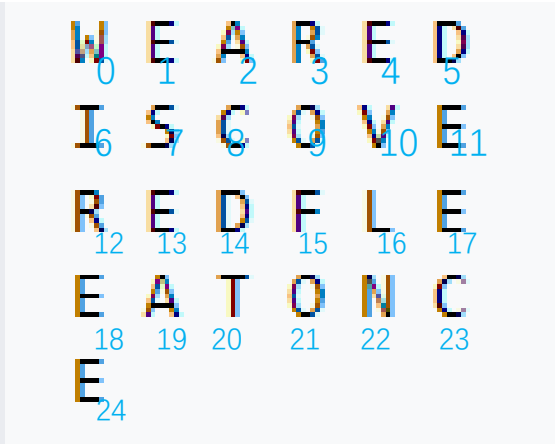
```
[4, 10, 16, 22, 2, 8, 14, 20, 1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 0, 6, 12, 18, 24]
```

If your code is correct, after running the above code, you'll get:

[4, 10, 16, 22, 2, 8, 14, 20, 1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 0, 6, 12, 18, 24]

What does it mean?

Index:	Z	E	B	R	A	S
	0	1	2	3	4	5



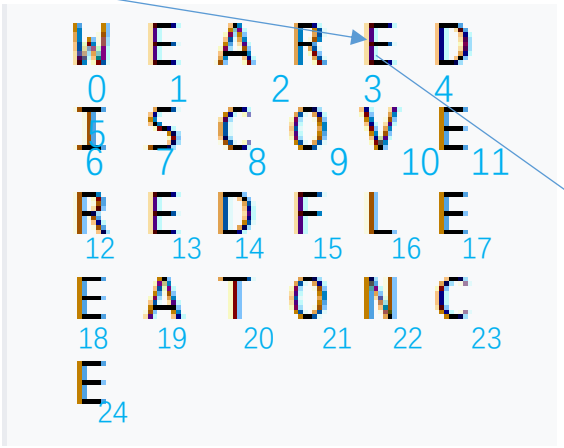
Based on columnar transposition cipher,
we first take out the letters in Column 'A' (index 4): E(4) V(10) L(16) N(22)
Then column 'B' (index 2): A(2) C(8) D(14) T(20)
Then column 'E' (index 1): E(1) S(7) E(13) A(19)
Then column 'R' (index 3): R(3) O(9) F(15) O(21)
Then column 'S' (index 5): D(5) E(11) E(17) C(23)
Then column 'Z' (index 0): W(0) I(6) R(12) E(18) E(24)

```
]: def ct_encrypt(plaintext, key):
       '''
       Return the ciphertext of a plaintext by the key using the irregular case
       of columnar transposition cipher.

       Parameters
       ----------
       plaintext: str
           a message in uppercase without punctuations/spaces.
       key: str
           secret key to encrypt plaintext.
       '''
       return ''.join([plaintext[i] for i in ct_idx(len(plaintext), key)])

key = 'ZEBRAS'
plaintext = 'WEAREDISCOVEREDFLEEATONCE'
ciphertext = 'EVLNACDTESEAROFODEECWIREE'
print(ct_encrypt(plaintext,key))
```

```
EVLNACDTESEAROFODEECWIREE
```

How to encrypt?
Pick the letters from the matrix based on their index and concatenate them together

[4, 10, 16, 22, 2, 8, 14, 20, 1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 0, 6, 12, 18, 24]



EVLNACDTESEAROFODEECWIREE

```
def ct_decrypt(ciphertext, key):
    '''
    Return the plaintext of the ciphertext by the key using the irregular case
    of columnar transposition cipher.

    Parameters
    ----------
    ciphertext (str): a string in uppercase without punctuations/spaces.
    key (str): secret key to decrypt ciphertext.
    '''

    #write your code here
```

Students you need to figure out by yourselves to get the mark.

The explanation in the slide should be enough for you to figure out the solution. It may take you some time beyond the lesson but is worthy and challenging.

How to decrypt?
Put the letters in ciphertext back based on their location

EVLNACDTESEAROFODEECWIREE

[4, 10, 16, 22, 2, 8, 14, 20, 1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 0, 6, 12, 18, 24]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Index: 0 1 2 3 4 5 6 7 8 9 10 …… 24

[W E A R E D I S C O V E R E D F L E E A T O N C E]

Then convert it to a string → 'WEAREDISCOVEREDFLEEATONCE'