# Practice Week 3

## Encryptions

# How to Process Files

- We open a file for reading
- We open a file for writing
- We read the file line by line and process all the data
- Then we write the processed data to another file
- Alternatively, we print out the results

# Today: Some simple cryptography

- Cryptography has been around for more than 2000 years
- Computers have had a big impact on cryptography and vice versa
- Here: Use two simple codes and break them
  - Take a text
  - Convert the text to only upper letters and suppress punctuation marks
  - Use the substitution code on the result

# How to Process a String

- Task:
  - Given a string, remove all punctuation marks and white spaces and make the text upper only
  - Hint: instead of typing the alphabet, you can use ascii_letters and digits

# How to Process a String

- We can import directly from string

```
from string import ascii_letters, digits
```

- We use the standard pattern:  Create a result array and add to it selectively

# How to Process a String

```
from string import ascii_letters, digits
def process(string):
    result = [ ]
    for letter in string:
        if letter in ascii_letters or letter in digits:
            result.append(letter.upper())
    return ''.join(result)
```

# How to Process a String

- This has an error.  Can you spot it?

```
from string import ascii_letters, digits

def process(string):
    result = [ ]
    for letter in string:
        if letter in ascii_letters or digits:
            result.append(letter.upper())
    return ''.join(result)
```

# How to Process a String

- This has an error.  Can you spot it?

```
from string import ascii_letters, digits

def process(string):
    result = [ ]
    for letter in string:
        if letter in ascii_letters or digits:
            result.append(letter.upper())
    return ''.join(result)
```

- digits is not zero and always true
- All letters are added to result

# How to Process a String

- This concatenates the letters and digits string and works

```
from string import ascii_letters, digits

def process(string):
    result = [ ]
    for letter in string:
        if letter in ascii_letters+digits:
            result.append(letter.upper())
    return ''.join(result)
```

# Substitution Cipher

- Replace every letter with another letter
- Classic use case for a dictionary
    - Generate a random dictionary that uses random.shuffle to shuffle a list
    - The upper letters are in strings.ascii_uppercase
    - Then create the translation dictionary

# Substitution Cipher

- To permute the list of letters:

```
let = list(ascii_uppercase)
random.shuffle(let)
```

# Substitution Cipher

- How to make a dictionary out of it?
  - Comprehension

```
substitution = { ascii_uppercase[i]:let[i] for i in
range(len(let))}
```

# Substitution Cipher

- How to make a dictionary out of it?
  - comprehension and zipping

```
substitution2 = { x[0]:x[1] for x in
zip(list(ascii_uppercase), let) }
```

# Substitution Cipher

- How to make a dictionary out of it?
  - Comprehension and enumerate

```
substitution3 = { letter:let[i] for i, letter in
enumerate(ascii_uppercase) }
```

# Substitution Cipher

- How to make a dictionary out of it?
  - Comprehension and using dict

```
substitution4 = dict(zip(ascii_uppercase, let))
```

# Substitution Cipher

- Using a key to generate the dictionary
  - Idea:  take a string "LOYOLA COLLEGE"
  - Then A->L, B->O, C->Y, D->?
    - Skip O and L
- D->A, E->E, F->G, and then?
- H -> next free letter:


- H->B, I->D, J->F, K->I, L->J, M->K, N->M, …

# Substitution Cipher

- How to solve this:
  - Make a list of possible choices by concatenating the list of letters in the key and all letters
  - ```
    ['L', 'O', 'Y', 'O', 'L', 'A', 'C', 'O',
    'L', 'L', 'E', 'G', 'E', 'A', 'B', 'C',
    'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
    'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
    'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
    ```
  - Assign the first letter to 'A'
    ```

# Substitution Cipher

- How to solve this:
  - Then remove this letter from the list:

```
[ele for ele in choices if ele != retval[letter]]
```

- ```
  ['O', 'Y', 'O', 'A', 'C', 'O',
  'E', 'G', 'E', 'A', 'B', 'C',
  'D', 'E', 'F', 'G', 'H', 'I',
  'J', 'K', 'M', 'N', 'O', 'P',
  'Q', 'R', 'S', 'T', 'U', 'V',
  'W', 'X', 'Y', 'Z']
  ```
  ```

# Substitution Cipher

- Complete code:

```
def make_dict(key):
    retval = { }
    choices = list(key)+list(ascii_uppercase)
    for letter in ascii_uppercase:
        print(choices)
        retval[letter] = choices[0]
        choices = [ele for ele in choices if ele !=
retval[letter]]
    return retval
```

# Substitution Cipher

- How to encode a whole book:
  - Write a function that takes a line and applies the dictionary to it
  - Write a function that opens the book for reading and an output-file for writing
    - Take all lines
    - Translate the lines
    - Print the translated line to the file

# Substitution Cipher

```
def translate(line, dictionary):
    result = [ ]
    for letter in line:
        if letter in digits:
            result.append(letter)
        else:
            result.append(dictionary[letter])
return ''.join(result)
```

# Substitution Cipher

```
def encode(infile, key):
    mydict = make_dict(key)
    with open(infile) as infile, open('output.txt','w') as outfile:
        for line in infile:
line = process(line)
new_line = translate(line, mydict)
print(new_line, file = outfile)
```

# Substitution Cipher

- How to decode?
  - Use the same procedure, but with the reversed dictionary

# Substitution Cipher

```
revdict = { mydict[letter]:letter for letter in mydict }
```

# Substitution Cipher

```
def decode(infile, outfile, key):
    mydict = make_dict(key)
    revdict = { mydict[letter]:letter for letter in mydict }
    with open(infile) as inputfile, open(outfile, 'w') as outputfile:
        for line in inputfile:
line = line.strip()
new_line = translate(line, revdict)
print(new_line, file = outputfile)
```

# Substitution Cipher

- How to break a substitution cipher?
  - Just do a frequency analysis
    - Open the encoded file and use a dictionary to count the letters

# Substitution Cipher

```
def get_frequencies(filename):
    count = {letter:0 for letter in ascii_uppercase}
    with open(filename) as infile:
        for line in infile:
            for letter in line:
                if letter in count:
                    count[letter]+=1
total = sum(list(count.values()))
for letter in ascii_uppercase:
    print(letter, ' : ', count[letter]/total*100)
```

# Substitution Cipher

- The frequency of letters is reasonably constant over texts in the same language

| Letter | Count | | Letter | Frequency |
|---|---|---|---|---|
| E | 21912 | | E | 12.02 |
| T | 16587 | | T | 9.10 |
| A | 14810 | | A | 8.12 |
| O | 14003 | | O | 7.68 |
| I | 13318 | | I | 7.31 |
| N | 12666 | | N | 6.95 |
| S | 11450 | | S | 6.28 |
| R | 10977 | | R | 6.02 |
| H | 10795 | | H | 5.92 |
| D | 7874 | | D | 4.32 |
| L | 7253 | | L | 3.98 |
| U | 5246 | | U | 2.88 |
| C | 4943 | | C | 2.71 |
| M | 4761 | | M | 2.61 |
| F | 4200 | | F | 2.30 |
| Y | 3853 | | Y | 2.11 |
| W | 3819 | | W | 2.09 |
| G | 3693 | | G | 2.03 |
| P | 3316 | | P | 1.82 |
| B | 2715 | | B | 1.49 |
| V | 2019 | | V | 1.11 |
| K | 1257 | | K | 0.69 |
| X | 315 | | X | 0.17 |
| Q | 205 | | Q | 0.11 |
| J | 188 | | J | 0.10 |
| Z | 128 | | Z | 0.07 |

# Substitution Cipher

```
A   :   1.3285085277595021
B   :   6.941140745989169
C   :   0.13411609899286 4
D   :   4.458727668404271
E   :   1.973784098385546
F   :   6.518548509539956
G   :   7.5408674528063155
H   :   1.3538134520977783
I   :   4.605496229566273
J   :   0.21509185687534796
K   :   5.159674072574523
L   :   1.9408876967457867
M   :   6.1415051368996405
N   :   8.522698517131435
O   :   12.725846449719116
P   :   2.3660104256288275
Q   :   9.671542082089173
R   :   1.0552153449061188
S   :   2.917657776203249
T   :   2.3584189483273446
U   :   6.807024646996306
V   :   0.751556252846804
W   :   2.495065539754036
X   :   0.11640265195607066
Y   :   1.8194240599220608
Z   :   0.08097575788248393
```

| Letter | Count | | Letter | Frequency |
|---|---|---|---|---|
| E | 21912 | | E | 12.02 |
| T | 16587 | | T | 9.10 |
| A | 14810 | | A | 8.12 |
| O | 14003 | | O | 7.68 |
| I | 13318 | | I | 7.31 |
| N | 12666 | | N | 6.95 |
| S | 11450 | | S | 6.28 |
| R | 10977 | | R | 6.02 |
| H | 10795 | | H | 5.92 |
| D | 7874 | | D | 4.32 |
| L | 7253 | | L | 3.98 |
| U | 5246 | | U | 2.88 |
| C | 4943 | | C | 2.71 |
| M | 4761 | | M | 2.61 |
| F | 4200 | | F | 2.30 |
| Y | 3853 | | Y | 2.11 |
| W | 3819 | | W | 2.09 |
| G | 3693 | | G | 2.03 |
| P | 3316 | | P | 1.82 |
| B | 2715 | | B | 1.49 |
| V | 2019 | | V | 1.11 |
| K | 1257 | | K | 0.69 |
| X | 315 | | X | 0.17 |
| Q | 205 | | Q | 0.11 |
| J | 188 | | J | 0.10 |
| Z | 128 | | Z | 0.07 |

# Caesar Cipher

- A simple form of a substitution cipher
  - Each letter is moved to the letter x spots away
    - Example:  x=5
      - A->F, B->G, C->H, D->I, …
        - When you reach 'Z', start with 'A' again:
      - U->Z, V->A, W->B, X->C, Y->D, Z->E

# Caesar Cipher

- We can use two built in functions that associate a letter with its encoding

```
>>> ord('य')
2351

>>> chr(2351)
'य'
```

# Caesar Cipher

- The encoding reflects the ordering of the alphabet.
  - To get the letter 5 steps afterwards:
    -
```
>>> def trans(character):
    return chr(ord(character)+5)

>>> trans('g')
'l'
>>> trans('य')
'ळ'
>>>
```

# Caesar Cipher

- The uppercase letters in the English alphabet are between ord('A') and ord('Z'), i.e. between 65 and 90
    - Implement the Caesar Cipher

# Caesar Cipher

```
def translate(letter, offset):
    number = ord(letter)-65
    new_number = number+offset
    if new_number > 25:
        new_number -=26
    return chr(new_number+65)
```