

# Cryptanalysis (암호분석)

Python Basic  
&  
Caesar Cipher Attack

2023.3

# Contents

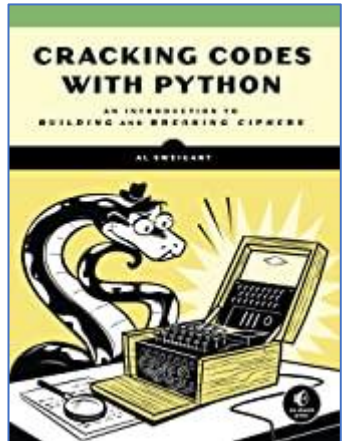
## ▶ Python 기초

- ▶ 사용자 정의 함수
- ▶ 함수의 파라미터 전달 방식
- ▶ 함수를 이용한 Caesar 암호 구현
- ▶ 파일 입출력
- ▶ 리스트(list)와 사전(dictionary) 타입

## ▶ Caesar 암호의 공격

- ▶ 영어사전 만들기
- ▶ 사전을 이용한 영문 판정법
- ▶ Caesar 암호의 공격 구현

**Cracking codes with Python:  
An Introduction to Building and Breaking Ciphers  
by Al Sweigart (2018)**



# 문자열(String) 사용 하기

- ▶ String concatenation with + operator
- ▶ String replication with \* operator
- ▶ Indexing with []
  - ▶ positive, negative index: [n] [-n]
- ▶ Slicing with [ : ] (range)
  - ▶ two indexes: [n:m]
  - ▶ blank index: [:m] [n:]
- ▶ Repeated index [:][:]

```
# 문자열 합치기  
print('Hello, ' + 'Python')
```

```
# 문자열 인덱싱  
print('Hello'[0])  
print('Hello'[-1])  
print('Hello'[-2])  
print('Hello'[2])
```

```
# 문자열 슬라이싱  
print('Hello, world'[0:4])  
print('Hello, world'[-5:-1])  
print('Hello, world'[-5:])
```

# 입출력 함수

- ▶ 출력 함수 print()
- ▶ 입력 함수 input()
- ▶ 주석(comments) #

```
# 사용자 입력 받기  
print("What is your name?")  
myName = input()  
print("Nice to meet you, " + myName + ".")
```

# Sample Code: Reverse Cipher

- ▶ 반복문 while
- ▶ 문자열 길이: len()

```
# 문자열 거꾸로 만들기
```

```
message = 'This is a sample text.'  
translated = ''  
i = len(message)-1  
while i >= 0:  
    translated = translated + message[i]  
    print('translated message = ', translated)  
    i = i - 1  
print('\n Final Result = ', translated)
```

# 문자열 검색 함수: find()

- ▶ 문자열 찾기: find()
  - ▶ 찾는 문자열의 인덱스(0~)
  - ▶ 찾지 못하면 -1
- ▶ 나머지 연산자 %
  - ▶ 연산의 우선 순위에 주의

```
#-- find()
msg = "abcdefghijklmnopqrstuvwxyz"
print(msg.find("def"))
print(msg.find("aa"))
print(msg[msg.find("j"):])
```

```
#--- % 연산자
print(5 % 3)
print( (10 + 10) % 3)
print( 10 + 10 % 3)
```

# Caesar Cipher (Encryption)

```
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
LowAlphabet = 'abcdefghijklmnopqrstuvwxyz'

plain_msg = 'This is a plaintext message to be encrypted.'
key = 3 # 암호키: select from (0-25)

##--- 암호화 과정
cipher_msg = ''
for symbol in plain_msg :
    if symbol in UpAlphabet:
        symbol_idx = UpAlphabet.find(symbol)
        trans_idx = (symbol_idx + key) % len(UpAlphabet)
        cipher_msg = cipher_msg + UpAlphabet[trans_idx]
    elif symbol in LowAlphabet:
        symbol_idx = LowAlphabet.find(symbol)
        trans_idx = (symbol_idx + key) % len(LowAlphabet)
        cipher_msg = cipher_msg + LowAlphabet[trans_idx]
    else:
        cipher_msg = cipher_msg + symbol

print('PLAINTEXT = ', plain_msg)
print('CIPHERTEXT = ', cipher_msg)
```

# Caesar Cipher (Decryption)

```
key = 3 # select from (0-25)

UpAlphabet    = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
LowerAlphabet = 'abcdefghijklmnopqrstuvwxyz'

#--- 복호화 과정
ciphertext = "wklv lv d sodlqwhaw phvvdjh wr eh hqfubswhg."
recovered_msg = ""

for symbol in ciphertext :
    if symbol in UpAlphabet:
        symbol_idx = UpAlphabet.find(symbol)
        trans_idx = (symbol_idx - key) % len(UpAlphabet)
        recovered_msg = recovered_msg + UpAlphabet[trans_idx]
    elif symbol in LowerAlphabet:
        symbol_idx = LowerAlphabet.find(symbol)
        trans_idx = (symbol_idx - key) % len(LowerAlphabet)
        recovered_msg = recovered_msg + LowerAlphabet[trans_idx]
    else:
        recovered_msg = recovered_msg + symbol
print('CIPHERTEXT = ', cipher_msg)
print('PLAINTEXT  = ', recovered_msg)
```



# 사용자 정의 함수

## ▶ 함수 정의 기본

```
def my_func(x,y):
```

```
    z = x+y
```

```
    return z
```

결과 값  
(return value)

```
a = 1
```

```
b = 2
```

```
print(my_func(a,b))
```

파라미터

```
def my_double(x,y):  
    return (2*x, 2*y)
```

```
a2, b2 = my_double(a, b)  
print(a2, b2)
```

# 함수 파라미터 전달 방식

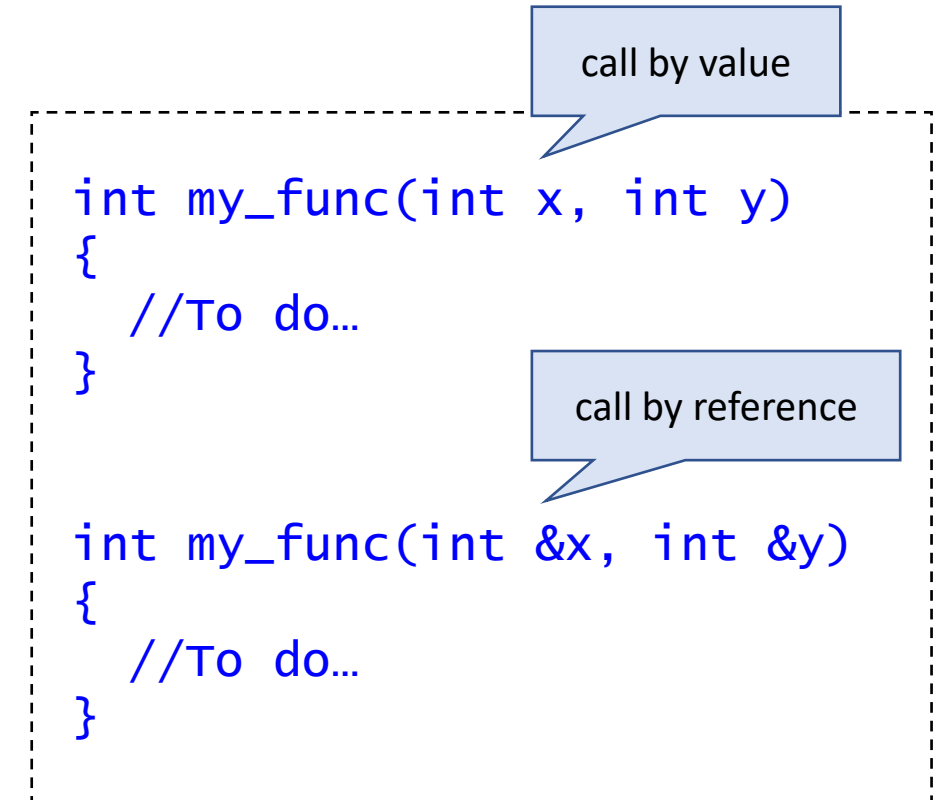
## ▶ C/C++의 파라미터 전달 방식

- ▶ Call by value
- ▶ Call by reference

## ▶ Python의 파라미터 전달 방식

- ▶ Call by object  
(=Call by object reference, =Call by sharing)

Python initially behaves like call-by-reference, but as soon as we are changing the value of such a variable, i.e., as soon as we assign a new object to it, Python "switches" to call-by-value.



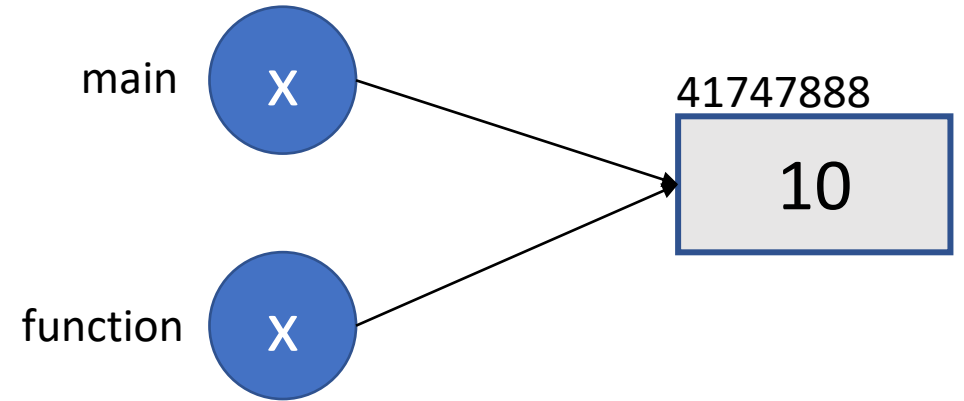
# Call by object

▶ 필요한 경우에만 복사본을 만든다.

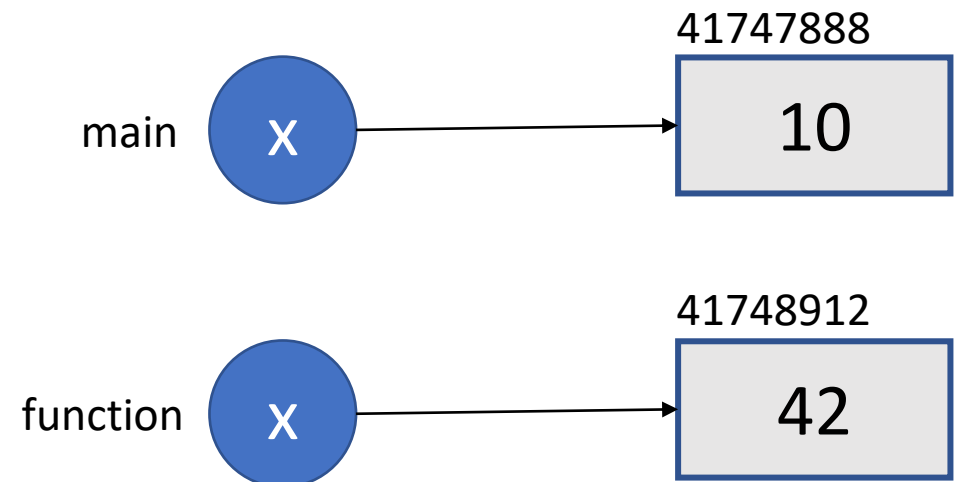
```
def ref_demo(x):  
    print("x=",x," id=",id(x))  
    x=42  
    print("x=",x," id=",id(x))
```

```
x = 10  
ref_demo(x)  
print("x=",x," id=",id(x))
```

x=42 이면?



x=42 실행 후



# In-place operation

## ▶ 함수로 전달된 파라미터의 값은 바뀔 수 있는가?

▶ Immutable variable vs Mutable variable

▶ In-place operation

immutable variable: 정수, 실수  
mutable variable: 리스트, 배열

```
def no_side_effects(cities):  
    print("cities=", cities, " id=", id(cities))  
    cities = cities + ["Birmingham", "Bradford"]  
    print("cities=", cities, " id=", id(cities))
```

```
locations = ["London", "Leeds", "Glasgow", "Sheffield"]  
no_side_effects(locations)  
print(locations)
```

```
def side_effects(cities):  
    print("cities=", cities, " id=", id(cities))  
    cities += ["Birmingham", "Bradford"] # in-place operation  
    print("cities=", cities, " id=", id(cities))
```

```
locations = ["London", "Leeds", "Glasgow", "Sheffield"]  
side_effects(locations)  
print(locations)
```

새로 메모리를  
할당하지 않고  
기존 데이터를  
업데이트 함

```
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield'] id= 2631971113800  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford'] id= 2631959055432  
['London', 'Leeds', 'Glasgow', 'Sheffield']  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield'] id= 2631959055432  
cities= ['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford'] id= 2631959055432  
['London', 'Leeds', 'Glasgow', 'Sheffield', 'Birmingham', 'Bradford']
```

# SWAP 함수

- ▶ swap() 두 변수의 값을 서로 바꾸는 함수

```
def my_swap(a,b):  
    print('a = ', a, 'b = ', b)  
    temp = a  
    a= b  
    b = temp  
    print('a = ', a, 'b = ', b)  
    return a,b
```

C언어 스타일로  
구현하면...

```
a, b = b, a
```

사실은  
swap() 함수를  
만들 필요 없음

# 함수를 이용한 Caesar Cipher 구현

## ▶ 함수로 정의된 암호화/복호화 사용

```
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
LowAlphabet = 'abcdefghijklmnopqrstuvwxyz'
```

전역변수

```
plain_msg = 'This is a plaintext message to be encrypted.'  
my_key = 13 # select from (0-25)  
cipher_msg = caesar_encrypt(my_key, plain_msg)  
print('PLAINTEXT = ', plain_msg)  
print('CIPHERTEXT = ', cipher_msg, '\n')
```

Caesar 암호를 함수로 정의하고  
필요할 때 호출하여 사용함

```
recovered_msg = caesar_decrypt(my_key, cipher_msg)  
print('CIPHERTEXT = ', cipher_msg)  
print('PLAINTEXT = ', recovered_msg)
```

# 함수를 이용한 Caesar Cipher 구현

## ▶ 암호화 함수

```
def caesar_encrypt(key, plain_msg):  
    cipher_msg = ''  
    for symbol in plain_msg :  
        if symbol in UpAlphabet:  
            symbol_idx = UpAlphabet.find(symbol)  
            trans_idx = (symbol_idx + key) % len(UpAlphabet)  
            cipher_msg = cipher_msg + UpAlphabet[trans_idx]  
        elif symbol in LowAlphabet:  
            symbol_idx = LowAlphabet.find(symbol)  
            trans_idx = (symbol_idx + key) % len(LowAlphabet)  
            cipher_msg = cipher_msg + LowAlphabet[trans_idx]  
        else:  
            cipher_msg = cipher_msg + symbol  
    return cipher_msg
```

# 함수를 이용한 Caesar Cipher 구현

## ▶ 복호화 함수

```
def caesar_decrypt(key, cipher_msg):  
    recovered_msg = ''  
    for symbol in cipher_msg :  
        if symbol in UpAlphabet:  
            symbol_idx = UpAlphabet.find(symbol)  
            trans_idx = (symbol_idx - key) % len(UpAlphabet)  
            recovered_msg = recovered_msg + UpAlphabet[trans_idx]  
        elif symbol in LowAlphabet:  
            symbol_idx = LowAlphabet.find(symbol)  
            trans_idx = (symbol_idx - key) % len(LowAlphabet)  
            recovered_msg = recovered_msg + LowAlphabet[trans_idx]  
        else:  
            recovered_msg = recovered_msg + symbol  
    return recovered_msg
```



# 출력 함수 print()의 포매팅

- ▶ C언어와 유사한 방법으로 print() 함수의 포맷이 가능함

```
n = 100
print('I trust you %s%%!' %(n))

Old_Name = 'Rijndael'
New_Name = 'AES'
print('%s is the old name of %s algorithm in 1990.' %(Old_Name, New_Name))
```

%를 출력할 땐 %%

주의!  
coma(,) 없음

# 파일 다루기

## ▶ 파일에서 읽어오기

```
import os, sys # 파일을 다루기 위한 라이브러리

in_file = 'my_text.txt'

if not os.path.exists(in_file):
    print('File %s does not exist.' %(in_file))
    sys.exit() # 프로그램 종료

#-- 입력파일에서 텍스트 읽기
InFileObj = open(in_file)
my_content = InFileObj.read()
InFileObj.close()

print(my_content)
```

```
# 작업 폴더 확인
print('Working directory : ',os.getcwd())

# 작업 폴더 변경
os.chdir('folder_name')
```

# 파일 다루기

## ▶ 파일에 쓰기

```
import os, sys # 파일을 다루기 위한 라이브러리

out_file = 'my_out.txt'

#-- 출력파일이 존재하면 덮어쓸지 물어보기
if os.path.exists(out_file):
    print('This will overwrite the file %s. (C)ontinue or (Q)uit' % (out_file))
    response = input('> ') # 사용자 입력 기다리기
    if not response.lower().startswith('c'):
        sys.exit()

outFileObj = open(out_file, 'w')
outFileObj.write(my_content)
outFileObj.close()
```

# 리스트 다루기

## ▶ 리스트

- ▶ (서로 다른 타입의) 데이터를 순서에 따라 모은 것

## ▶ 리스트 만들기, 인덱싱

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
#-- 리스트 인덱싱
```

```
print(animals[0])
```

```
print(animals[1:])
```

```
print(animals.index('man'))
```

리스트에 없으면  
오류가 발생한다!

# 리스트 다루기

- ▶ 원소 추가 방법
- ▶ 원소 여부를 확인하기

```
animals += 'man'
```



```
animals += list('man')
```

```
list('man') # ['m', 'a', 'n']
```

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
#-- Append (리스트, 문자열, 메소드)
```

```
animals += ['man']
```

```
animals += 'man'
```

```
animals.append('woman')
```

```
print(animals)
```

```
if 'man' in animals:
```

```
    print('A man is an animal.')
```

3가지 방법은 모두 같은 결과?

원소 확인

# 리스트 다루기

- ▶ 리스트를 이용한 반복문
- ▶ 리스트를 문자열로

```
animals = ['cat', 'dog', 'lion', 'tiger', 'hippo', 'snake', 'bird']
```

```
for pet in animals:  
    print('I have a %s.' % (pet))
```

```
#-- 리스트 -> 문자열 (join)  
print (''.join(animals))
```

리스트를 문자열로 변환

```
print(3*[1,2,3]+[9])
```

이 결과는?

# Dictionary 다루기

## ▶ Dictionary 데이터 타입

- ▶ Dictionary: (key, value) 의 모임
- ▶ 숫자 인덱스를 사용하지 않고 키(key)에 대응되는 값(value)를 저장함
- ▶ 해시함수를 이용하여 원소(key, value)에 빠른 접근이 가능(순차적 검색아님)

```
myDic1 = { 'us' : 'AES', 'kr' : 'LEA', 'jp' : 'MISTY' }  
print(myDic1['kr'])
```

```
#-- copy  
myDic2 = myDic1  
myDic2['ru'] = 'GOST'  
print(myDic1)  
print(myDic2)
```

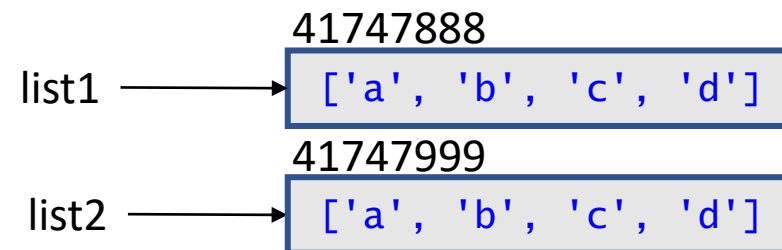
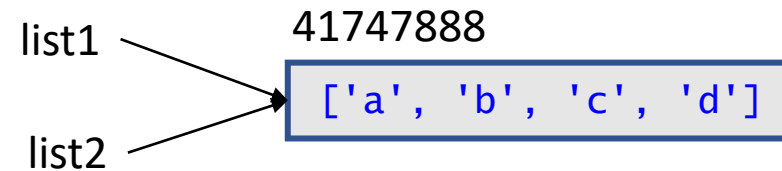
myDic2[0]과 같은  
접근은 가능하지 않음!

두 Dictionary가 같아진다!

# Shallow and Deep Copy

## ▶ 복잡한 구조의 복사 방법

- ▶ Shallow copy: 같은 메모리를 참조
- ▶ Deep copy: 새로운 메모리 할당



```
#shallow copy
```

```
list1 = ['a', 'b', 'c', 'd']  
list2 = list1 #shallow copy  
list2[0] = 'A'  
print('list1 = ', list1)  
print('list2 = ', list2)
```

```
#deep copy (copy lib)
```

```
import copy  
list1 = ['a', 'b', 'c', 'd']  
list2 = copy.deepcopy(list1) #deep  
copy  
list2[0] = 'A'  
print('list1 = ', list1)  
print('list2 = ', list2)
```

deepcopy를 위한  
라이브러리



# Deep Copy and Slice Operator

## ▶ 리스트의 깊은 복사

- ▶ Slice operator를 이용한 깊은 복사: 1단계까지만 가능 (아래 예제를 확인!!!)
- ▶ Deep copy(copy library): 모든 단계의 깊은 복사

```
#deep copy (copy lib)
```

```
list1 = ['a', 'b', ['c', 'd']]
list2 = copy.deepcopy(list1)
list2[2] = 'CD'
list3 = copy.deepcopy(list1)
list3[2][0] = 'C'
print('list1 = ', list1)
print('list2 = ', list2)
print('list3 = ', list3)
```

```
#deep copy (slice operator)
```

```
list1 = ['a', 'b', ['c', 'd']]
list2 = list1[:] #slice operator
list2[2] = 'CD'
list3 = list1[:]
list3[2][0] = 'C'
print('list1 = ', list1)
print('list2 = ', list2)
print('list3 = ', list3)
```

# Split과 Join

- ▶ split: 문자열을 나누어 리스트로
- ▶ join: 리스트를 문자열로

```
msg = 'This is a sample text'  
list_msg = msg.split()  
print('msg = ', msg)  
print('list = ', list_msg)
```

공백을 기준으로 나눈다.

```
joined_msg = ''.join(list_msg)  
print('joined = ', joined_msg)
```

공백없는 문자열로 합쳐진다.

```
for k in range(len(list_msg)-1):  
    list_msg[k] += ' '  
joined_msg2 = ''.join(list_msg)  
print('joined2 = ', joined_msg2)
```

원래 문자열과 같은 결과

# 영어 사전 활용하기

## ▶ 영어단어사전 활용함수

- ▶ dictionary.txt: 영어단어로 된 파일
  - ▶ 영어 단어로된 dictionary 데이터 만들기

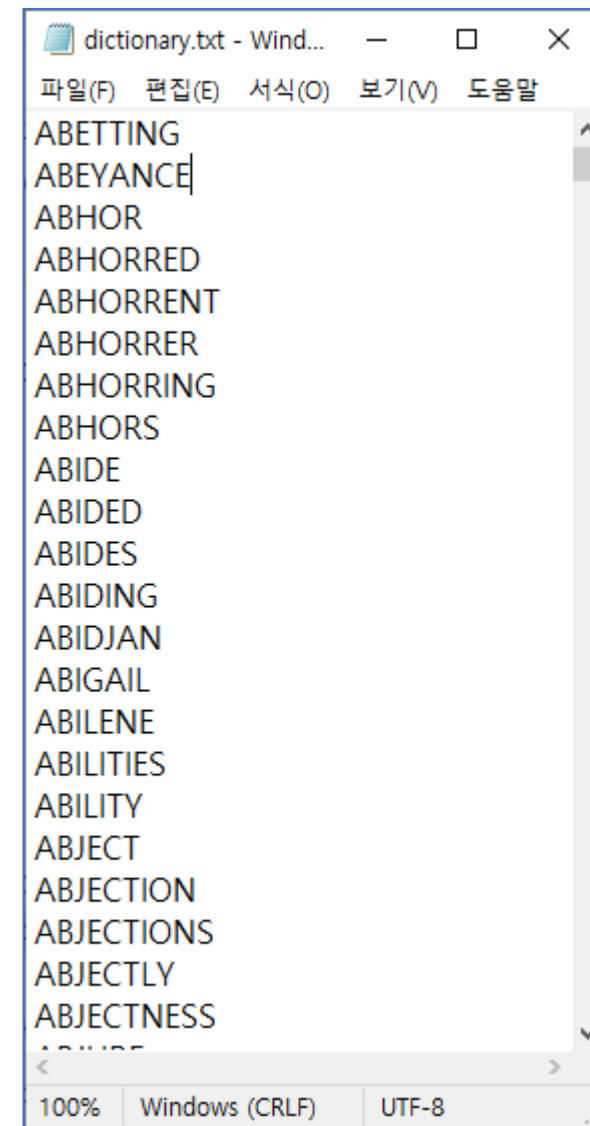
```
def loadDictionary():  
    dictionary_file = open('dictionary.txt')  
    EnglishWords = {}  
    for word in dictionary_file.read().split('\n'):  
        EnglishWords[word] = None  
    dictionary_file.close()  
    return EnglishWords
```

-- 전역변수

```
EnglishWords = loadDictionary()
```

(key, value) → (word, None)  
예: { 'this': None, 'is' : None }

- ▶ 주어진 텍스트가 영어인지 판정하는 함수 만들기
  - ▶ isEnglish('This is a sample') → True



# 영어단어 다루기

## ▶ removeNonLetters()

- ▶ 문자열에서 영문자, 공백만 남기기

```
letters_and_space = UpAlphabet + UpAlphabet.lower() + '\t\n'
```

```
#---- 특수문자, 숫자 지우기
def removeNonLetters(message):
    letters_only = []
    for ch in message:
        if ch in letters_and_space:
            letters_only.append(ch)
    return ''.join(letters_only)
```

```
#-- 전역변수
EnglishWords = loadDictionary()
```

## • percentEnglishWords()

- 영어사전에 있는 단어의 비율

```
#---- 올바른 영어단어의 비율
def percentEnglishWords(message):
    message = message.upper()
    message = removeNonLetters(message)
    possible_words = message.split()

    if possible_words == []:
        return 0.0
    count_words = 0
    for word in possible_words:
        if word in EnglishWords:
            count_words += 1
    return
float(count_words)/len(possible_words)
```

0으로 나누는  
오류발생을 방지

영어사전에 있는  
단어인지?

# 영어 판정 함수: isEnglish()

- ▶ 주어진 문자열이 영어로 된 것인지 판정하는 함수
  - ▶ 복호화 분장이 바르게 되었는지 판정할 때 사용

```
#--- 영어인지 판정하기
def isEnglish(message, wordPercentage=20, letterPercentage=80):
    wordsMatch = percentEnglishWords(message)*100 >= wordPercentage

    numLetters = len(removeNonLetters(message))
    messageLettersPercentage = float(numLetters) / len(message) * 100
    lettersMatch = messageLettersPercentage >= letterPercentage

    return wordsMatch and lettersMatch
```

영어 단어의 비율이  
충분함

영문자의 비율이  
충분함

# Caesar Cipher Attack 2

## ▶ 영어의 특성을 이용한 Caesar Cipher 공격법

```
import CaesarCipher_lib
import EngDic_lib
import os, sys
```

```
ciphertext = 'Znoy oy g ygsvrk'
print('CIPHERTEXT = ', ciphertext)
```

```
for key in range(0,26):
    recovered_msg = CaesarCipher_lib.caesar_decrypt(key, ciphertext)
    PercentEngWords = EngDic_lib.percentEnglishWords(recovered_msg)*100
    print('key # %2s : %s (English word: %5.1f%%)' % (key, recovered_msg, PercentEngWords) )
```

키 전수조사

이 값이 가장 큰 경우가  
올바른 암호키 !!!

```
CIPHERTEXT = Znoy oy g ygsvrk
key # 0 : Znoy oy g ygsvrk (English word: 0.0%)
key # 1 : Ymnx nx f xfruqj (English word: 0.0%)
key # 2 : Xlmw mw e weqtpi (English word: 0.0%)
key # 3 : Wklv lv d vdpsoh (English word: 0.0%)
key # 4 : Vjku ku c ucorng (English word: 0.0%)
key # 5 : Uijt jt b thngmf (English word: 0.0%)
key # 6 : This is a sample (English word: 75.0%)
key # 7 : Sghr hr z rlekd (English word: 0.0%)
key # 8 : Rfgq gq y qyknjc (English word: 0.0%)
key # 9 : Qefp fp x pxjmib (English word: 0.0%)
key #10 : Pdeo eo w owilha (English word: 0.0%)
key #11 : Ocdn dn v nvhkgz (English word: 0.0%)
key #12 : Nbcm cm u mugjfy (English word: 0.0%)
key #13 : Mabl bl t ltfiex (English word: 0.0%)
key #14 : Lzak ak s ksehdw (English word: 0.0%)
key #15 : Kyzj zj r jrdgcv (English word: 0.0%)
key #16 : Jxyi yi q iqcfbu (English word: 0.0%)
key #17 : Iwxh xh p hpbeat (English word: 0.0%)
key #18 : Hvwg wg o goadz (English word: 0.0%)
key #19 : Guvf vf n fnzcyr (English word: 0.0%)
key #20 : Ftue ue m emybxq (English word: 0.0%)
key #21 : Estd td l dlxawp (English word: 0.0%)
key #22 : Drsc sc k ckwzvo (English word: 0.0%)
key #23 : Cqrb rb j bjvyun (English word: 0.0%)
key #24 : Bpqa qa i aiuxtm (English word: 0.0%)
key #25 : Aopz pz h zhtwsl (English word: 0.0%)
```