1 파이썬 내장 함수 설명표

파이썬에서 지원하는 내장 함수명이나 내장 클래스명을 변수명으로 사용하면 해당 객체를 사용할 수 없게 되므로, 내장 객체의 이름을 변수명으로 사용하지 않도록 주의하자. 즉, 아래 표에 있는 내장 함수명들은 변수명으로 부적절하다.

함수명	한글 설명	사용 예
abs()	숫자의 절댓값을 반환한다.	>>> abs(-100) 100
all()	반복 가능한 객체의 모든 아이템들이 True이면, True를 반환한다.	>>> all([True, True, True]) True
any()	반복 가능한 객체의 원소 중에 하나라도 True가 있으면, True를 반환한다.	>>> any([True, False, False]) True
ascii()	객체를 읽을 수 있는 버전으로 반환한다. 아스키(Ascii)[1] 문자가 아닌 경우에는 이스케이프 문자를 써서 변경한다.	>>> ascii("Mötley Crüe") '"M\xf6tley Cr\xfce"'
bin()	입력받은 숫자를 이진값을 반환한다.	>>> bin(127) '0b1111111'
bool()	특정 객체에 대한 불리언값을 반환한다.	>>> bool(1) True
bytearray()	바이트의 배열을 반환한다.	>>> bytearray('파이썬', 'cp949') bytearray(b'\xc6\xc4\xc0\xcc\xbd\xe3')
bytes()	바이트 객체를 반환한다.	>>> bytes('파이썬', 'euc-kr') b'\xc6\xc4\xc0\xcc\xbd\xe3'
callable()	입력받은 객체를 호출할 수 있으면 True를 반환한다.	>>> callable(object) True
chr()	유니코드 숫자를 문자로	>>> chr(65)

[1] 미국 정보 교환 표준 부호(American standard code for information interchange, ASCII)를 뜻하는 아스키는 영문 알파벳을 표시하는 대표적인 문자 인코딩이다. 아스키는 7비트 인코딩으로 총 128개의 문자로 이루어져 있으며, 이중 95개는 출력이 가능하고 나머지 33개는 출력이 불가능한 제어 문자이다.

	반환한다.	'A'

classmethod()	메서드를 클래스 메서드로 변경한다.	>>> class MyClass:
		  @classmethod
		  def add(cls, x, y):
		  return x + y
		>>> MyClass.add(1, 2)
		3

compile()	특정 코드를 실행 가능한 객체로 반환한다.	>>> code = compile("name='Wick, J.'\nprint(name)", "test", "exec")
		>>> exec(code)
		Wick, J.

complex()	복소수를 반환한다.	>>> complex(1, 2) (1+2j)

delattr()	특정 객체의 어트리뷰트(프로퍼티 또는 메서드)를 제거한다.	>>> class MyClass:
		  prop = 1
		>>> delattr(MyClass, 'prop')
		>>> dir(MyClass)
		['__class__', '__delattr__', '__dict__', '__dir__', 중략, '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']

dict()	딕셔너리를 반환한다.	>>> dict({'A':65, 'B':66, 'C':67}) {'A': 65, 'B': 66, 'C': 67}
dir()	특정 객체의 프로퍼티와 메서드들의 리스트를 반환한다.	>>> dir(str) ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', 중략, 'title', 'translate', 'upper', 'zfill']
divmod()	첫 번째 인자를 두 번째 인자로 나누었을 때의 값과 나머지를 반환한다.	>>> divmod(5, 3) (1, 2)
enumerate()	튜플 등의 컬렉션 객체를 인자로 받아서 열거형 객체를 반환한다.	>>> [x for x in enumerate(['a', 'b', 'c'])] [(0, 'a'), (1, 'b'), (2, 'c')]
eval()	표현식을 평가하고 실행한다.	>>> x = 1; eval('x + 1') 2
exec()	특정 코드나 객체를 실행한다.	>>> exec('n = 1 + 2') >>> n

		3

filter()	필터 함수를 사용하여 반복 가능한 객체에서 아이템들을 제외한다.	>>> def is_even(x): return x % 2 == 0
		>>> f = filter(is_even, [1, 2, 3, 4, 5])
		>>> print(*f)
		2 4

float()	부동소수점수을 반환한다.	>>> float('-3.14') -3.14
format()	입력값의 형식을 정한다.	>>> format(123456789, ',') '123,456,789'

frozenset()	변경 불가능한 형태의 프로즌셋 객체를 반환한다. 프로즌셋은 셋과 유사하나 값을 변경하지 않는 연산만 허용한다.	>>> fs = frozenset([1, 2, 3])
		>>> fs
		frozenset({1, 2, 3})

getattr()	특정 어트리뷰트(프로퍼티와 메서드)의 값을 반환한다.	>>> import math
		>>> getattr(math, 'pi')
		3.141592653589793

globals()	현재의 글로벌 심볼 테이블을 딕셔너리 형태로 반환한다.	>>> globals() {'__name__': '__main__', '__doc__': None, '__package__': None, 중략, '__builtins__': <module 'builtins' (built-in)>}
hasattr()	특정 객체가 특정 어트리뷰를 가지고 있다면 True를 반환한다.	>>> hasattr(str, 'isalpha') True

hash()	특정 객체의 해시값을 반환한다. 동일한 값(value)을 가지고 있는 객체는 동일한 해시값을 가진다.	>>> hash(1)
		1
		>>> hash(1.0)
		1

help()	빌트-인 헬프 시스템을 실행한다.	>>> help(hash) Help on built-in function hash in module builtins: hash(obj, /) Return the hash value for the given object. Two objects that compare equal must also have the same hash value, but the reverse is not necessarily true.

hex()	숫자를 16진수로 변환한다.	>>> hex(16) '0x10'
id()	객체의 ID를 반환한다.	>>> id('string') 3039952641920
input()	표준입력으로 사용자로부터 값을 입력받는다.	>>> age = input("Input your age : ") Input your age : 20
int()	정수 형태의 숫자를 반환한다.	>>> int(0x10) 16

isinstance()	입력된 객체가 인스턴스이면 True를 반환한다.	>>> s = str('string')
		>>> isinstance(s, str)
		True

issubclass()	입력된 클래스가 특정 객체의 하위 클래스이면 True를 반환한다.	>>> issubclass(str, object) True

iter()	이터레이터 객체를 반환한다.	>>> it = iter([1, 2, 3])
		>>> next(it); next(it); next(it)
		1
		2
		3

len()	객체의 길이를 반환한다.	>>> len(['a', 'b', 'c']) 3
list()	리스트를 반환한다.	>>> list(range(3)) [0, 1, 2]
locals()	현재의 로컬 심볼 테이블을 딕셔너리로 반환한다.	>>> locals() {'__name__': '__main__', '__doc__': None, '__package__': None, 중략, '__builtins__': <module 'builtins' (built-in)>}

map()	반복 가능한 객체를 특정 함수에 매핑한 결과를 반환한다.	>>> def cube(x): return x * x * x
		>>> m = map(cube, [1, 2, 3, 4, 5])
		>>> print(*m)
		1 8 27 64 125

max()	반복 가능한 객체의 가장 큰 아이템을 반환한다.	>>> max([1, 2, 3]) 3

memoryview()	메모리 뷰 객체를 반환한다.	>>> memoryview(b'ABC')[0] 65
min()	반복 가능한 객체의 가장 작은 아이템을 반환한다.	>>> min(['A', 'B', 'C']) 'A'

next()	반복 가능한 객체의 다음 아이템을 반환한다.	>>> it = iter('ABC')
		>>> next(it)
		'A'

object()	새로운 객체를 반환한다.	>>> object().__doc__ 'The most base type'
oct()	숫자를 8진수로 변환한다.	>>> oct(8) '0o10'
open()	파일을 열어서 파일 객체를 반환한다.	>>> open("C:\read_me.txt", "rb").read() b'Hello!'
ord()	특정 문자에 대한 유니코드 숫자를 반환한다.	>>> ord('A') 65
pow()	첫 번째 인자의 수를 두 번째 인자의 수 만큼 거듭제곱한 결과를 반환한다.	>>> pow(2, 8) 256
print()	표준 출력 장치로 표시한다.	>>> print("□□□□□", end="\r"); print("▌▌") ▌▌□□□

property()	프로퍼티를 가져오거나 정하거나 삭제한다.	>>> class Person:
		  def __init__(self, name):
		  self._name = name
		  @property
		  def name(self):
		  print('Getting name')
		  return self._name
		>>> p = Person("Wick, J.")
		>>> p.name
		Getting name
		'Wick, J.'

range()	0부터 1씩 증가하는 숫자들을 반환한다.	>>> [x for x in range(10)] [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

repr()	객체의 공식 문자열 표현을 반환한다.	>>> class Test:
		  def __repr__self):
		  return 'the "official" string representation of an object'

		>>> cls = Test()
		>>> repr(cls)
		'the "official" string representation of an object'

reversed()	거꾸로 정렬된 이터레이터를 반환한다.	>>> it = reversed([1, 2, 3, 4, 5])
		>>> print(*it)
		5 4 3 2 1

round()	반올림한 숫자를 반환한다.	>>> round(3.141592653589793, 2) 3.14
set()	새로운 셋 객체를 반환한다.	>>> set([1, 2, 2, 3, 3, 3]) {1, 2, 3}

setattr()	객체의 어트리뷰트(프로퍼티나 메서드)를 설정한다.	>>> class Person:
		  age = 10
		>>> p = Person()
		>>> setattr(p, 'age', 20)
		>>> p.age
		20

slice()	객체의 일부만 잘라서 반환한다.	>>> li = ['a', 'b', 'c', 'd', 'e']
		>>> x = slice(0, 5, 2)
		>>> li[x]
		['a', 'c', 'e']

sorted()	정렬된 리스트를 반환한다.	>>> sorted([1, 3, 5, 4, 2]) [1, 2, 3, 4, 5]

@staticmethod()	메서드를 스태틱 메서드로 변환한다.	>>> class MyClass:
		  @staticmethod
		  def add(x, y):
		  return x + y
		>>> MyClass.add(1, 2)
		3

str()	문자열 객체를 반환한다.	>>> str('Spam') 'Spam'
sum()	이터레이터의 아이템들의 합을 구한다.	>>> sum([1, 2, 3]) 6

super()	부모 또는 형제 클래스의 메서드를 호출하기 위한 프록시 객체를 반환한다.	>>> class A:
		  def method(self):
		  print('A')
		>>> class B(A):
		  def method(self):

		print('B') print(super().method()) >>> B().method() B A None

tuple()	튜플을 반환한다.	>>> tp = tuple([1, 2, 3])
		>>> tp
		(1, 2, 3)

type()	입력 객체의 타입을 반환한다.	>>> type(0xdeadc0de)
vars()	객체의 __dict__ 프로퍼티를 반환한다.	>>> vars() {'__name__': '__main__', '__doc__': None, '__package__': None, 중략, '__builtins__': <module 'builtins' (built-in)>}
zip()	둘 이상의 이터레이터를 인자로 받아서 하나의 이터레이터로 반환한다.	>> z = zip([1, 2, 3], ['one', 'two', 'three']) >>> print(*z) (1, 'one') (2, 'two') (3, 'three')

2 파이썬 AES-256 암복호화

최근 들어 프로그램의 취약점을 노리는 해킹 기법이 늘어남에 따라, 프로그래머들도 보안성과 관련하여 많은 고민을 해야하는 시기가 왔다. 특히 암호화 모듈은 필수적으로 사용해야 하므로, 파이썬을 이용해 대표적인 암호화 알고리즘인 AES를 사용하는 방법에 대해 알아보자.

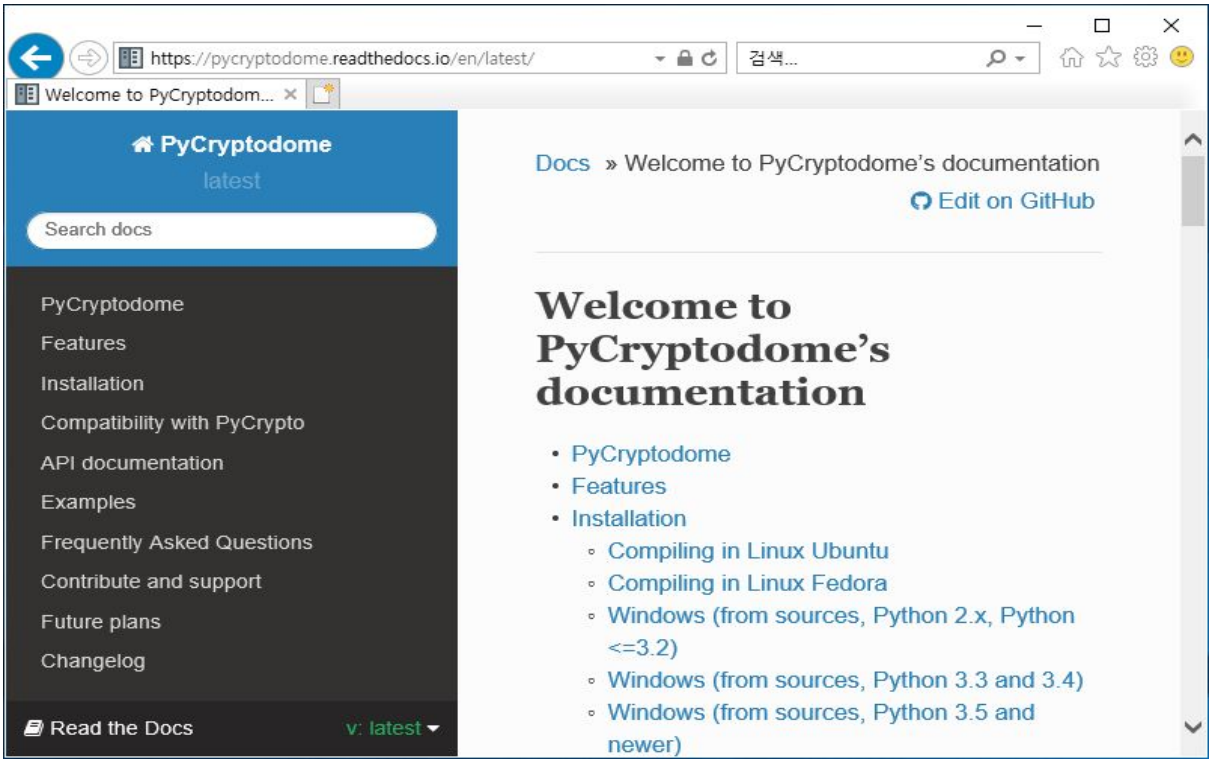
2.1 파이크립토돔 패키지

몇 년 전까지만 해도 파이썬으로 AES 암호화를 하기 위해서는 파이크립토(PyCrypto) 패키지를 주로 이용하였다. 하지만 지금은 업데이트가 중단된지 오래되었기 때문에, 최신 버전의 파이썬이 설치된 환경에서 파이크립토를 설치하기가 매우 어렵다. 따라서 최근에는 파이크립토돔이라는 패키지를 주로 이용한다.

2.1.1 파이크립토돔 설치

파이크립토돔(PyCryptodome)은 파이크립토의 마지막 공식 버전인 2.6.1에서 갈라진(fork) 패키지이며, 기존에 파이크립토를 설치했던 사용자를 위해두 가지 설치 방식을 제공한다. pip install pycryptodome으로 설치할 경우, 기존의 파이크립토를 파이크립토돔으로 교체하는 방식으로 설치하며 모든 모듈은 Crypto 패키지에 포함된다. pip install pycryptodomex로 설치할 경우, 기존의 파이크립토와 별도로 파이크립토돔을 설치하며 신규 모듈은 Cryptodome 패키지에 포함된다.

파이크립토돔 공식 홈페이지(https://pycryptodome.readthedocs.io)에는 패키지 사용법과 여러 암호화 알고리즘에 대한 문서화가 잘 되어있으니 방문해서 읽어보자. 파이크립토돔을 설치한 적이 없다면 먼저 pip install pycryptodome으로 파이크립토돔부터 설치하자.



2.2 AES-128 (EAX 방식)

AES(Advanced Encryption Standard)는 미국 국립 표준기술 연구소(NIST)에 의해 표준화된 대칭형 블록 암호화 알고리즘이다. 블록 암호화 방식이기 때문에 암호화 대상 데이터는 16바이트(byte)의 배수이어야 한다. AES는 빠르고 안전하기 때문에 대칭형 암호화 방식에서 사실상의 표준이라고 볼 수 있다.

1바이트가 8비트(bit)이므로 키가 16바이트일 경우, 키의 길이는 16 * 8비트 = 128비트가 되며, 이를 AES-128이라고 한다. 마찬가지로 키가 24바이트이면 AES-192, 키가 32바이트이면 AES-256으로 구분한다.

우리가 암호화 할 문장은 'AES is a symmetric block cipher standardized by NIST.'이다. AES 모듈은 키와 데이터 인자를 바이트 스트링(byte string)으로 받기 때문에, 키와 데이터 모두 문자열 앞에 b를 붙여서 바이트 스트링으로 선언한다.

```
from Crypto.Cipher import AES

key = b'16ByteKey4AES128'
plain_text = b'AES is a symmetric block cipher standardized by NIST.'

print("plain_text.decode('utf-8') : {}".format(plain_text.decode('utf-8')))
print('plain_text.hex()  : {}'.format(plain_text.hex()))
```

바이트 한 글자는 8비트 숫자 범위(0~255)에 해당하기 때문에 7비트 ASCII 문자(0~127)로는 전부 표현할 수 없다. 따라서, 바이트 문자를 표시하기 위해서는 일반적으로 UTF-8 형식으로 디코딩을 한다. hex() 함수를 사용할 경우, 바이트 한 글자를 16진수 두 글자로 나타낼 수 있다. 즉, 아래 예제에서 첫 바이트인 A는 16진수 41(0x41)로 표시되는 것을 확인할 수 있다.

<실행결과>

```
plain_text.decode('utf-8') : AES is a symmetric block cipher standardized by
NIST.
plain_text.hex()  :
414553206973206120737796d6d657472696320626c6f636b206369706865722207374616e64617264
697a6564206279204e4953542e
```

</실행결과>

2.2.1 AES-128 (EAX 방식) 파일 암호화

EAX(encrypt-then-authenticate-then-translate) 모드는 블록 암호 운영 방식의 하나로 AEAD(Authenticated Encryption with Associated Data)로 표현하기도 한다. 수신자는 메시지 인증 코드(Message Authentication Code) 태그를 이용해 메시지가 변조되었는지 검증할 수 있다.

EAX 모드의 AES 객체는 논스(nonce, number used once)라고 불리는 일회용 숫자값을 가지고 있다. 논스는 임의로 생성되는 암호화 토큰으로 볼 수 있으며 재생 공격을 방지하기 위해 사용된다. 복호화를 하려면 암호화할 때 사용했던 키와 논스가 필요하다.

```
from Crypto.Cipher import AES

key = b'16ByteKey4AES128'
```

```
plain_text = b'AES is a symmetric block cipher standardized by NIST.'

aes_enc = AES.new(key, AES.MODE_EAX)
aes_nonce = aesEnc.nonce
cipher_text = aes_enc.encrypt(plain_text)
print("cipher_text.decode('latin') : {}".format(cipher_text.decode('latin')))
print('cipher_text.hex() : {}'.format(cipher_text.hex()))

with open('encrypted.bin', 'wb') as f:
  f.write(cipher_text)
```

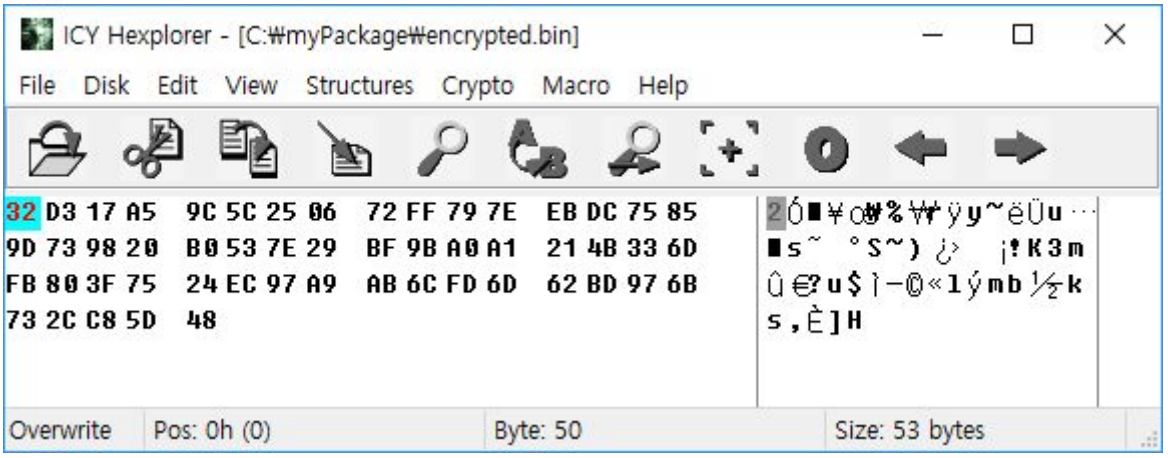
b'AES is a symmetric block cipher standardized by NIST.'를 암호화한 ciper_text의 결과는 아래와 같다. ciper_text는 바이트 스트링이므로 문자열 형태로 보기 위해서는 decode() 함수를 사용하여 출력해야 한다. hex() 함수를 사용하면 16진수 형태로 변환할 수 있다. 아래의 두 값은 동일한 바이트 스트링을 나타낸다.

<실행결과>

```
cipher_text.decode('latin') : 2Ó¥\%rÿy~ëÜus °S~)¿ ¡!K3mû?u$ì(c)«lýmb1/2ks,È]H
cipher_text.hex() :
32d317a59c5c250672ff797eebdc75859d739820b0537e29bf9ba0a1214b336dfb803f7524ec97a9
ab6cfd6d62bd976b732cc85d48
```

</실행결과>

cyber_text의 내용이 encrypted.bin 파일에 제대로 쓰여졌는지 확인하기 위해 hex 에디터 프로그램으로 encrypted.bin 파일을 열어서 비교해보자. 바이트 스트링을 문자열로 디코딩 하는 과정에서 일부 문자들이 약간 다르게 보이긴 하지만, 서로 같은 바이트 스트링이라는 것을 알 수 있다.



2.2.2 AES-128 (EAX 방식) 복호화

encrypted.bin 파일을 읽어서 복호화하는 코드는 아래와 같이 간단하다.

```
with open('encrypted.bin', 'rb') as f:
  enc_text = f.read()

aes_dec = AES.new(key, AES.MODE_EAX, aes_nonce)
```

```
dec_text = aes_dec.decrypt(enc_text)
print("dec_text.decode('utf-8') : {}".format(dec_text.decode('utf-8')))
```

암호화 할 때 사용했던 키(key)와 일회용 숫자(aes_nonce)로 AES 객체를 생성한 후 decrypt() 함수를 호출하면 아래와 같이 복호화된 평문을 얻을 수 있다.

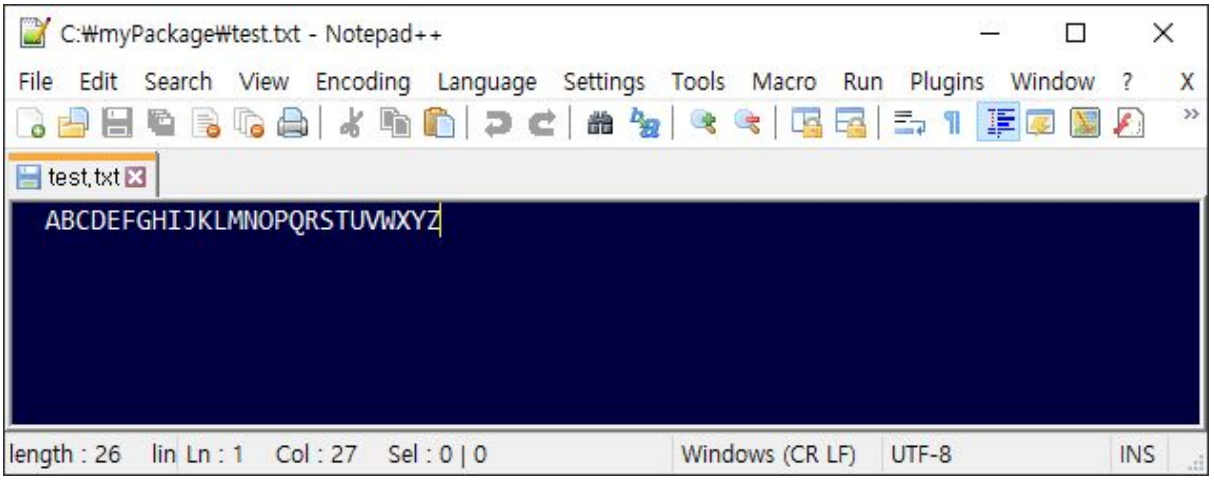
<실행결과/>

```
dec_text.decode('utf-8') : AES is a symmetric block cipher standardized by NIST.
```

</실행결과>

2.3 AES-256 (CBC 방식)

CBC(Ciphertext Block Chaining) 방식에서 각 블록은 암호화되기 이전 블록의 암호화 결과와 XOR되기 때문에 암호 블록 체인 방식이라고도 부른다. AES-256 CBC 방식으로 암호화할 파일을 Notedpad++ 등을 이용해서 생성하자. A~Z까지 알파벳 26자를 입력한 뒤 test.txt 파일로 저장한다.



2.3.1 AES-256 암/복호화 32 바이트 키 생성

AES-256으로 암/복호화하기 위해서는 32바이트 키가 필요하다. 32바이트 길이의 키는 너무 길어 불편하니 외우기 쉬운 단어로 SHA-256 해시값을 생성하여 32바이트 키를 만들어보자. 파이크립토돔의 SHA-256을 이용해 32바이트 해시를 생성하는 법은 아래와 같다.

```
>>> from Crypto.Hash import SHA256
>>> data = b'myPa$$w0rd'
>>> SHA256.new(data).hexdigest()
'f9fd3bad9c8a2701909cc511b077e2508894dbb4f3679dd6077377fe7f18be07'
```

2.3.2 초기화 벡터

초기화 벡터(Initialization Vector, IV)는 암호화 과정에서 첫 블록을 암호화 할 때 사용되는 값이다. 초기화 벡터는 해커에게 노출되어도 무방하지만 초기화 벡터를 반복해서 사용하는 것은 위험하다. 동일한 초기화 벡터를 사용하면 암호화 결과가 같아지기 때문에 매 암호화마다 다른 초기화 벡터를

사용해야 한다.

16바이트 길이의 초기화 벡터는 Crypto.Random 패키지의 get_random_bytes() 함수로 간단히 생성할 수 있다.

```
>>> from Crypto.Random import get_random_bytes
>>> iv = get_random_bytes(16)
>>> print(iv.hex())
91e5ad5eb818e8bcef73cffd890b46ab
```

파일 크기는 os 표준 라이브러리를 사용하여 아래와 같이 구한다.

```
import os
fileSize = os.path.getsize('test.txt')
```

2.3.3 블록 크기에 맞추어 패딩 하기

CBC 방식에서 암호화 대상은 길이가 16의 배수인 블록이다. Crypto.Util.Padding의 pad() 함수를 이용하여 알파벳 26자에 패딩 문자(\x06) 6자를 더해서 16의 배수인 32바이트 길이로 만들 수 있다.

```
>>> from Crypto.Util.Padding import pad
>>> pad(b'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 16)
b'ABCDEFGHIJKLMNOPQRSTUVWXYZ\x06\x06\x06\x06\x06\x06'
```

2.3.4 스트럭트 모듈로 파일 크기 기록하기

스트럭트(struct) 모듈은 C 언어의 구조체 형식과 파이썬 언어의 자료형 간의 변환을 위해 사용하는 모듈이다. CBC 암호화 예제에서는 목적지 파일 내부에 파일 크기를 기록하기 위해서 pack() 함수를 사용할 것이다. 파일에 기록된 숫자는 읽는 방식에 따라 그 값이 달라지기 때문에, 정해진 방식에 따라 기록하고 정해진 방식에 따라 읽어야 한다.

text.txt 파일의 크기를 나타내는 숫자 26(=\x1A)을 리틀 엔디언 방식(<)으로 unsigned long(L) 길이인 4바이트의 물리적 디스크에 기록하면, 실제로 기록되는 값은 '\x1A \x00 \x00 \x00'이다.

```
>>> import struct
>>> fileSize = struct.pack('<L', 26)
>>> fileSize
b'\x1a\x00\x00\x00'

>>> struct.unpack('<L', fileSize)
(26,)
```

리틀 엔디언 방식(<)의 unsigned long(L) 형식이 몇 바이트인지 계산하려면 calcsize() 함수를 이용하면 된다.

```
>>> struct.calcsize('<L')
4
```

문자	바이트 순서	크기	정렬
@	시스템에 따름	시스템에 따름	시스템에 따름
=	시스템에 따름	표준	없음
<	리틀-엔디언	표준	없음
>	빅-엔디언	표준	없음
!	네트워크(= 빅-엔디언)	표준	없음

포맷 문자의 첫 문자가 위의 문자가 아닐 경우 기본값인 '@'이 설정되었다고 가정한다. 바이트 순서란 메모리 상에 바이트를 배열하는 순서를 의미하며 CPU마다 다른데, Intel x86과 AMD64 (x86-64)는 리틀-엔디언을 사용하고, Motorola 68000과 PowerPC G5는 빅-엔디언을 사용한다.

빅-엔디언의 경우 사람이 숫자를 읽고 쓰는 방법과 같기 때문에 디버깅 과정에서 메모리의 값을 볼 때 편하다. 예를 들어 0x12345678은 빅-엔디언으로는 12 34 56 78로 표현되고 리틀-엔디언으로는 78 56 34 12로 표현된다.

반면에 리틀-엔디언은 메모리에 저장된 값에서 하위 비트를 추출해서 사용할 때 편리하다는 장점이 있다. 예를 들어 16진수 0x1A를 리틀-엔디언으로 표현하면 1A 00 00 00이 되는데, 하위 16비트를 구하려면 별도의 계산 없이 메모리 시작 주소에서부터 16비트만 읽어들이면 된다.

ARM과 Intel Itanium은 바이-엔디언(Bi-endian)으로 빅-엔디언과 리틀-엔디언을 선택해서 사용할 수 있도록 설계되었다. 참고로, 엔디언(endian)이라는 단어는 조너선 스위프트(Jonathan Swift)의 '걸리버 여행기(Gulliver's Travels)'에서 소인국 사람들이 달걀을 먹을 때 뭉툭한 부분을 먼저 깨는 파(빅-엔디언)와 뾰족한 부분을 먼저 깨는 파(리틀-엔디언)로 나뉘어 싸우는 장면에서 유래되었다.

파이썬으로 시스템의 엔디언을 체크하려면 sys.byteorder를 확인한다.

```
>>> import sys
>>> sys.byteorder
'little'
```

포맷 문자	C 언어 자료형	파이썬 자료형	표준 크기
X	pad byte	값 없음	없음
C	char	길이 1인 bytes	1
B	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1

H	short	integer	2
H	unsigned short	integer	2
I	int	integer	4
I (대문자 i)	unsigned int	integer	4
I (소문자 L)	long	integer	4
L	unsigned long	integer	4
Q	long long	integer	8
Q	unsigned long long	integer	8
F	float	float	4
D	double	float	8
S	char[]	bytes	없음
P	void *	integer	없음

포맷 문자 앞에 정수를 붙이면 해당 포맷 문자를 정수만큼 반복한 것과 동일하다. 예를 들어 포맷 스트링 '4h'는 'hhhh'와 동일하다.

2.3.5 AES-256 파일 암호화 함수 구현

AES-256 파일 암호화 기능을 담당하는 encryptFile() 함수는 아래와 같이 구현할 수 있다.

```
from Crypto.Cipher import AES
from Crypto.Hash import SHA256
from Crypto.Util.Padding import pad
from Crypto.Random import get_random_bytes
import os, struct

def encrypt_file(key, srcfile, dstfile=None):
    """ Encrypts a file using AES-256 (CBC mode) with the given key.

        key:
            byte string to hash
            SHA-256 will make 32 bytes long hashed key for AES-256.

        srcfile:
```

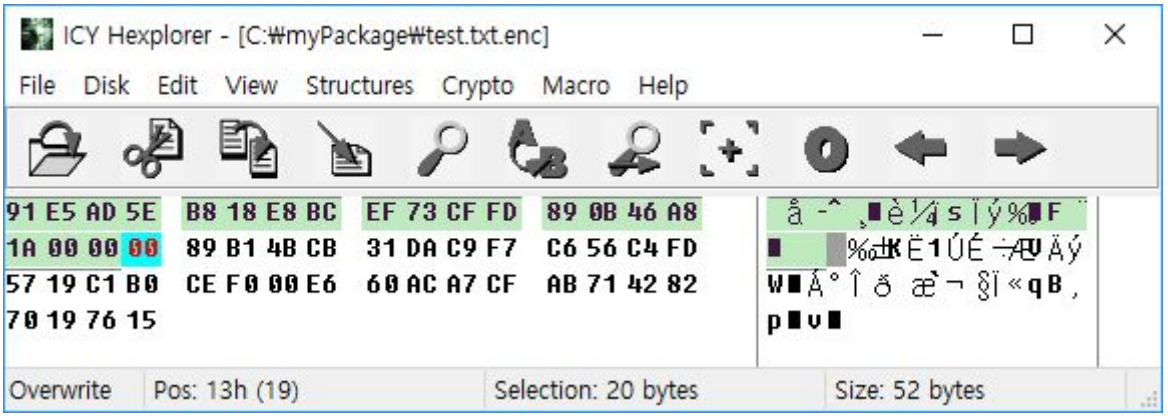
```
            Name of the source file

        dstfile:
            Name of the destination file
            If None, 'srcfile.enc' will be used.
    """
    if not dstfile:
        dstfile = srcfile + '.enc'

    BUF_SIZE = 1024 * 16 # Buffer size must be divisible by 16
    fileSize = os.path.getsize(srcfile)
    iv = get_random_bytes(16)
    hashed_key = SHA256.new(key).digest()
    aes = AES.new(hashed_key, AES.MODE_CBC, iv)

    with open(srcfile, 'rb') as sf, open(dstfile, 'wb') as df:
        df.write(iv)
        df.write(struct.pack('<L', fileSize))
        while True:
            buffer = sf.read(BUF_SIZE)
            if len(buffer) == 0:
                break
            elif len(buffer) % 16 != 0:
                buffer = pad(buffer, 16)
            df.write(aes.encrypt(buffer))
```

2.3.6 AES-256 암호화 결과 확인하기

아래와 같이 encrypt_file() 함수를 이용해 test.txt 파일을 암호화하면 test.txt.enc 파일이 생성된다.

```
encrypt_file(b'myPa$$w0rd', 'test.txt')
```

암호화된 test.txt.enc 파일을 hex 에디터로 열어보면 아래와 같다. hex 에디터에서 선택 표시된 영역은 20 Bytes로, 초기화 벡터 16 Bytes와 파일 크기 4 Bytes를 포함한다.



암호화된 파일 구조 (총 56 Bytes)

초기화 벡터(IV)	파일크기	원본 텍스트에 대한 암호화	패딩 문자에 대한 암호화
16 Bytes	4 Bytes (1A 00 00 00 = 26)	26 Bytes	6 Bytes (42 82 70 19 76 15)

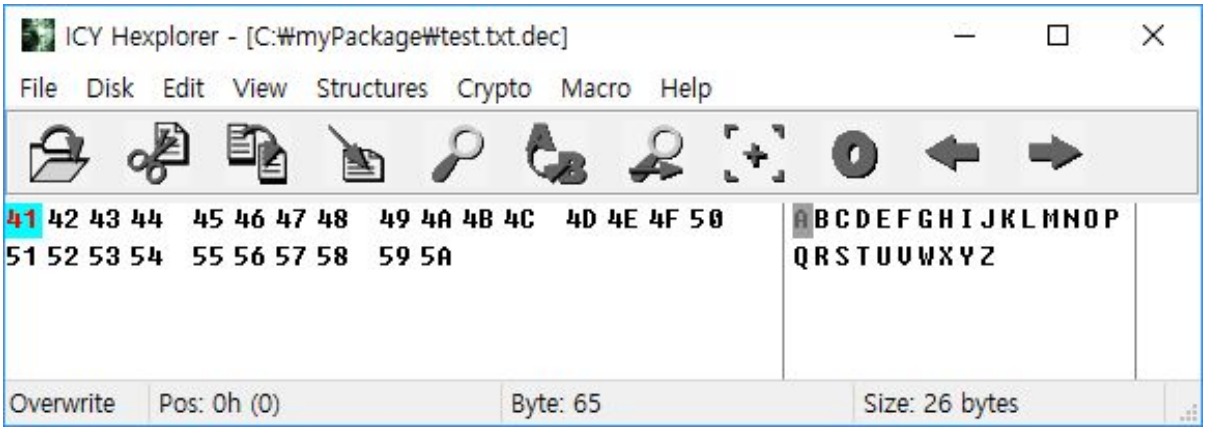
2.3.7 AES-256 복호화 함수 구현

AES-256 파일 복호화 기능을 담당하는 decrypt_file() 함수는 아래와 같이 구현한다.

```
def decrypt_file(key, srcfile, dstfile=None):
    """ Decrypts a file using AES-256 (CBC mode) with the given key.

        key:
            byte string to hash
            SHA-256 will make 32 bytes long hashed key for AES-256.

        srcfile:
            Name of the source file

        dstfile:
            Name of the destination file
            If None, '<srcfile>.dec' will be used.
    """
    if not dstFile:
        dstfile = srcfile.rstrip('.enc') +'.dec'

    BUF_SIZE = 1024 * 16
    with open(srcfile, 'rb') as sf, open(dstfile, 'wb') as df:
        iv = sf.read(16)
        org_filesize = struct.unpack('<L', sf.read(struct.calcsize('L')))[0]
        hashed_key = SHA256.new(key).digest()
        aes = AES.new(hashed_key, AES.MODE_CBC, iv)
        while True:
            buffer = sf.read(BUF_SIZE)
            if len(buffer) == 0:
                break
            df.write(aes.decrypt(buffer))
        df.truncate(org_filesize)
```

2.3.8 AES-256 복호화 결과 확인하기

```
decryptFile(b'myPa$$w0rd', 'test.txt.enc')
```



2.3.9 윈도우 fc 명령으로 파일 비교하기

os.system() 함수를 사용하면 파이썬 셸에서도 윈도우 명령을 사용할 수 있다. FC(File Compare) 명령을 이용해 원본 텍스트 파일과 복호화한 파일이 동일한지 확인해보자.

<문법>

```
os.system("fc 파일명1 파일명2")
```

</문법>