CSCI-SHU 210 Data Structures

Assignment 1 Python Review and OOP

Problem 1 Block-wise Palindromes

A number is block-wise palindrome if it reads the same backward or forward **in blocks size of 2**. Given a **positive** integer *n*. Our goal is to check if the **binary** representation of *n* is a block-wise palindrome or not.

Implement function block_palindrome(n) that figures out whether an integer *n* is a block-wise palindrome in its binary representation. Return True if it is, and False otherwise.

Example 1: 215 = $(11\ 01\ 01\ 11)_b$ so it is a block-wise palindrome in blocks size of 2.

```
Input: block_palindrome(215)
Return: True
```

Example 2: 38 = $(10\ 01\ 10)_b$ so it is a block-wise palindrome in blocks size of 2.

```
Input: block_palindrome(38)
Return: True
```

Example 3: 153 = $(10\ 01\ 10\ 01)_b$ so it is not a block-wise palindrome in blocks size of 2.

```
Input: block_palindrome(153)
Return: False
```

Requirements:

- You should do it without casting n into str/list/etc. (do not use bin() either).
- Moreover, you need to use **only bitwise operators**, such as &, |, >>, <<, Xor, etc.
- You can also allow yourself to use =, >, < for conditional expressions, and +, - for basic counting.
- If *n* has an odd number of digits in binary representation, **add a 0** before its binary representation to make it even digits. Except for this 0, **no other leading 0's** are being considered (see e.g.:)

Example 4: 105 = $(1\ 10\ 10\ 01)_b$, since it has odd numbers of digits, 105 =$(01\ 10\ 10\ 01)_b$. So it is a block-wise palindrome in blocks size of 2.

```
Input: block_palindrome(105)
Return: True
```

Problem 2 String Generator

Write a Python generator that yields all possible strings formed by using the characters ‘c’ , ‘a’ , ‘t’ , ‘d’ , ‘o’ , and ‘g’ exactly once.

That is, you should define a generator string_generator(). When called, it generates an iterator:

Example 1:

catdog_it = string_generator()
Input: next(catdog_it), next(catdog_it), next(catdog_it), …… next(catdog_it), ……
Return: ‘godtac’, ‘godtca’, ‘godatc’, ……, ‘tadcgo’, ……

Requirements:

- The method has to be non-recursive
- The order in which your generator yields strings does not matter

Problem 3: Dr X Cipher

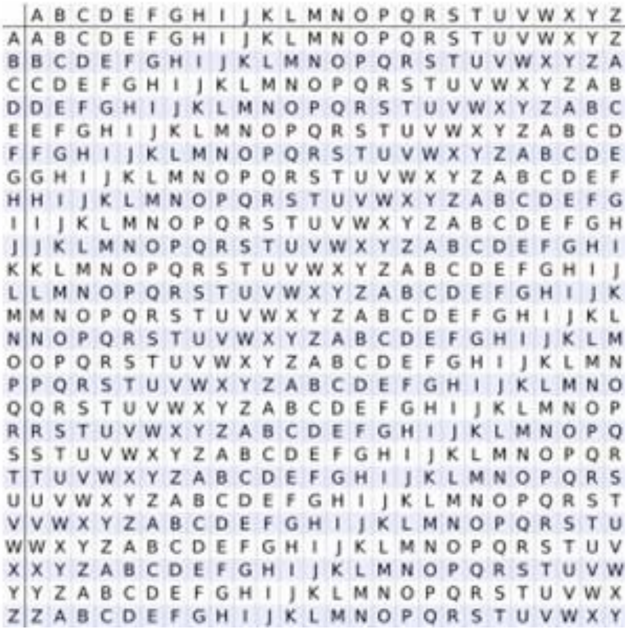
Dr X comes up with an interesting way of encrypting messages. The algorithm uses a text-key to encrypt the plain text, and the key is constructed partially from the plain text. The algorithm is demonstrated in the following image.

Dr X Cipher

Plain Text: ATT ACK ATD AWN
Key: QUE
Filled-up key: QUE **TTA TTA TTA**
Ciphertext: QNX TVK TMD TPN

Dr X Decipher

Ciphertext: QNX TVK TMD TPN
Key: QUE
Filled-up key: ??? ??? ??? ???
Original Text: ATT ACK ATD AWN



	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

If the key is long enough, Dr X simply uses the table on the right. If the key is shorter than the plaintext, Dr X uses the plaintext to fit to the proper length. Suppose the key is of length *j* < len(plaintext). Then we use the j^{th}, $(j-1)^{th}$, $(j-2)^{th}$…, 1^{st} characters from the plaintext to fit in. If one fit is not enough, we pad the j^{th}, $(j-1)^{th}$, $(j-2)^{th}$…, 1^{st} characters again.

Implement functions, encryptX(plain, key), decryptX(cipher, key)

encryptX(plain, key) takes two parameters, the plain text string and the key string. It should return the corresponding cipher text.

decryptX(cipher, key) takes two parameters, the cipher text string and the keystring. It should return the corresponding plain text.

Important:

- You can assume all strings are in upper case.
- Think about how to determine the key when deciphering (the ??? above).
- Hint: use ord(), chr() functions

Example 1:

Input: encryptX(“ATTACKATDAWN”, “QUE”)
Return: QNXTVKTMDTPN

Example 2:

Input: decryptX(“QNXTVKTMDTPN”, “QUE”)
Return: ATTACKATDAWN

Problem 4: Object oriented programming

In this question, we are going to implement the MyComplex class, representing complex numbers in the form of “a+b*i*”.

MyComplex
+ field: real_part, use a float number + field: imaginary_part, use a float number
+ method: __add__(other) + method: __iadd__(other) +method: __sub__(other) +method: __mul__(other) + method: __eq__(other) + method: __truediv__(other) + method: __str__()

Important:

- The starting point for this problem is provided in the assignment.
- Support the following operations:
  - MyComplex + MyComplex
  - MyComplex += MyComplex
  - MyComplex – MyComplex
  - MyComplex * MyComplex
  - MyComplex / MyComplex
  - MyComplex == MyComplex
  - print(MyComplex)
- You can define new functions and call your new functions. Just make sure the original provided test code runs without problem.
- Recall: $(a+bi) / (c+di) = (a+bi)*(c-di) / (c^2+d^2)$
- Check your implementation’s correctness with the given test code