CS 3430: SciComp with Py
Assignment 3
Encrypting and Decrypting Strings

Vladimir Kulyukin
Department of Computer Science
Utah State University

January 28, 2017

1 Learning Objectives

1. Strings
2. Pascal's Triangle
3. Encryption
4. Decryption
5. Multi-module Python programs

2 Introduction

In this assignment, you will write Py2 and Py3 functions to encrypt and decrypt texts. You will use the functions you implemented in Assignment 2 such as `half_interval_method` and `euclid_number`.

3 Pascal's Triangle

The encryption method you will implement in this assignment will utilize Pascal's Triangle, a pattern of numbers named after the French mathematician Blaise Pascal who investigated some properties of this infinite pattern in his "Treatise of Arithmetic Triangle" written in 1653. The name Pascal's triangle is common only in the West. The triangle was known to the Indian grammarian and mathematican Pingala in the 2nd century B.C. Incidentally, Pingala also discovered the pattern of numbers known in the West as the Fibonacci sequence. The sequence is named after the Italian mathematician Leonardo Fibonacci who described it in his "Book of Calculation" in 1202. Another noteworthy fact about Pascal's triangle is that in Iran and, more broadly, in Persian mathematical culture, the triangle is called the Khayyam triangle after the great Persian mathematician, poet, and scholar Omar Khayyam who proved several theorems about the triangle's binomial coefficients in the early 12th century, well before Blaise Pascal. Naming conventions work in mysterious ways, and History does have a strange sense of humor when She assigns credit!

Below are the first 17 rows of Pascal's Triangle from 0 upto 16. The leftmost number followed by the colon specifies the row number. The other integers are the row's elements. As you can see, each row begins and ends with 1. Each entry in row n and column k denotes $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. For example, the top of the triangle, row 0, is $1 = \binom{0}{0} = \frac{0!}{0!(0-0)!}$. The entries for row 1 are 1 and 1, and can be computed as follows: $\binom{1}{0} = \frac{1!}{0!(1-0)!} = 1$ and $\binom{1}{1} = \frac{1!}{1!(1-1)!} = 1$. The entries for row 2 are 1, 2, and 1, and can be computed as follows: $\binom{2}{0} = \frac{2!}{0!(2-0)!} = 1$, $\binom{2}{1} = \frac{2!}{1!(2-1)!} = 2$, and $\binom{2}{2} = \frac{2!}{2!(2-2)!} = 1$.

```
0:  1
1:  1 1
2:  1 2 1
3:  1 3 3 1
4:  1 4 6 4 1
5:  1 5 10 10 5 1
6:  1 6 15 20 15 6 1
7:  1 7 21 35 35 21 7 1
8:  1 8 28 56 70 56 28 8 1
9:  1 9 36 84 126 126 84 36 9 1
10: 1 10 45 120 210 252 210 120 45 10 1
```

```
11: 1 11 55 165 330 462 462 330 165 55 11 1
12: 1 12 66 220 495 792 924 792 495 220 66 12 1
13: 1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
14: 1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
15: 1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
16: 1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
```

Pascal's triangle is full of many amazing properties. One such property is that if n is a prime number, then all elements in row n, except the 1's on both ends, are divisible by n. For example, for $n = 3$, 3 divides all elements in row 3, except the outer 1's. The same is true for 5, 7, 11, 13, etc.

Let us implement a few functions that will be used in the encryption process below. First, implement the function `pascal_tri_row(n)` that computes all entries in row n of Pascal's Triangle:

Below are a few trial runs to guide your implementation.

```
>>> pascal_tri_row(3)
[1, 3, 3, 1]
>>> pascal_tri_row(5)
[1, 5, 10, 10, 5, 1]
>>> pascal_tri_row(7)
[1, 7, 21, 35, 35, 21, 7, 1]
>>> pascal_tri_row(11)
[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

And a few of the 212 elements in row 211, the 3rd Euclid number.

```
>>> pascal_tri_row(euclid_number(3))
[1,
211,
22155,
1543465,
80260180,
3322771452,
114081819852,
3340967581380,
85194673325190,
1921613187223730,
38816586381919346,
709284896615071686,
11821414943584528100L,
180958582597947776300L,
2559271382456689979100L,
33611764156264528392180L,
411744110914240472804205L,
4722947154604523070401175L,
50902874888515415314323775L,
517066044920182902929709925L,
4963834031233755868125215280L,
45147252379316541467234100880L,
389908088730461039944294507600L,
3204027337828571154324854866800L,
25098214146323807375544696456600L,
...
]
```

The next step after `pascal_tri_now(n)` is to implement the function `pascal_tri_row_sum(n)`. If we let $R_n = pascal\_tri\_row\_sum(n)$, then `pascal_tri_row_sum(n)` is defined in equation (1).

$$pascal\_tri\_row\_sum(n) = \begin{cases} 2 + \sum_{i \in R_n, i \neq 1} (\frac{i}{n}), & \text{if } n \text{ is prime} \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

A few test runs are below.

```
>>> pascal_tri_row_sum(3)
4
>>> pascal_tri_row_sum(5)
8
>>> pascal_tri_row_sum(7)
20
>>> pascal_tri_row_sum(euclid_number(3))
1559719959546166864601866523751043132685057692543677595591O988L
```

4 Encryption and Decryption

Everything is now in place for us to tackle string encryption and decryption. The intellectual roots of encryption go as far back as the Pythagorean, Chaldean, and Kabbalah systems of numerology. At the heart of these systems is the assumption, impossible to prove but frequently observable, that there is a deep correlation between numbers and the laws governing the universe. Many scholars throughout history have used number patterns to encode and decode hidden meanings into and out of various texts.

Cryptography distinguishes two types of keys: public and private. Both keys are typically, but not always. are numbers, big numbers. The public key is available for everyone to see. The private key is computed from the public key and is privately kept for encrypting and decrypting texts. There are two types of private keys: static and dynamic. A static key is computed once and kept on some storage device or a piece of paper for encryption and decryption purposes. A dynamic key is an algorithm that runs on a CPU to encrypt and decrypt texts by extracting some features from a public key. So long as nobody knows how that algorithm works and which public keys it uses, the encryption and decryption are safe. You will implement a dynamic encryption/decryption solution, i.e., an algorithm that uses a public key to encrypt and decrypt messages.

Typically both static and dynamic solutions use prime factors of a public key to compute private keys. Why prime factors? Because the larger the number the harder it gets to compute its prime factors. So the first step is to compute the prime factors of a public key. Let us keep it simple for now and assume that our public key is 210. The prime factors of 210 are 2, 3, 5, and 7. Let us fix these values programmatically.

```
public_key = 210
public_key_factors = (2, 3, 5, 7)
```

Another element we need for encryption is a character separator. If we want to decrypt encrypted texts we need to know where one encrypted character ends and another begins. We will use euclid numbers as separators. For now, let us use the 4th euclid number, 2311, as our separator and store it in a variable.

```
code_separator = euclid_number(4)
```

Randomization is another trick frequently done to make encryption stronger. Toward that end, let us define the function `choose_random_factor_index(num_public_key_factors)`.

```
from random import randint
def choose_random_factor_index(num_public_key_factors):
    return randint(0, num_public_key_factors-1)
```

This function allows us to randomly choose a prime factor index as follows.

```
>>> i = choose_random_factor_index(len(public_key_factors))
>>> i
0
>>> public_key_factors[i]
2
>>> i = choose_random_factor_index(len(public_key_factors))
>>> i
2
>>> public_key_factors[i]
5
```

Now we can outline the algorithm for encrypting a character that you will implement in the function `encrypt_char(c)`.

1. Let c be a character we need to encode.
2. Let pi be a random index returned by `choose_random_factor_index`.

3. Let $p =$ `public_key_factors[pi]`.
4. Let $s =$ `public_tri_row_sum(p)`.
5. Let z be the floor of the zero of $(x^9 - s + x)/2$, $0 < x < 1000$ found with the half interval method.
6. The character encoding is the concatentation of the string representations of pi, $code\_separator$, the sum of $\lfloor z \rfloor$ and $ord(c)$, and $code\_separator$.

Let us do a few test runs.

```
>>> encrypt_char('a')
'32311982311'
>>> encrypt_char('b')
'22311992311'
>>> encrypt_char('c')
'123111002311'
```

Let us take a closer clook at the first encryption, i.e., '32311982311'. The first element, 3, is a random index computed in step 2. It is followed by the code separator 2311. The prime factor computed in step 3 is 7, i.e., `public_key_factors[3]`. The value of s computed in step 4 is `pascal_tri_row_sum(7)` $= 20$. Solving $(x^9 - s + x)/2$, $0 < x < 1000$, where $s = 20$, by the half interval method gives us $z = 1.3838811495725558$ in step 5. For step 6, we need to compute $\lfloor z \rfloor + ord('a') = 1 + 97 = 98$. Finally, in step 6 we compute the contatenation of the string representations of the computed values: $'3' +' 2311' +' 98' +' 2311' =' 32311982311'$. The encryptions of 'b' and 'c' are computed similarly.

To encrypt a string is to concatenate the encryptions of all individual characters in that string in the order of their occurrence from left to right. It should be noted that one can also go from right to left or from the middle to both ends. But we will do it from left to right.

```
def encrypt_text(txt):
    enc = ''
    for c in txt: enc += encrypt_char(c)
    return en
```

A couple of test runs:

```
>>> encrypt_text('abc')
'1231198231132311992311123111002311'
>>> encrypt_text('efgh,!@')
'12311102231122311103231122311104231102311105231122311452311323113423110231165231l'
```

Now implement the function `decrypt_text(encrypted_text)`. This function uses the code separator to find all the character encodings and reverses the encryption steps to recover the encrypted character. A few test runs are below.

```
>>> enc = encrypt_text('abc')
>>> enc
'0231198231122311992311023111002311'
>>> decrypt_text(enc)
'abc'
>>> enc = encrypt_text('efgh,!@')
>>> enc
'12311102231122311103231122311104231122311105231102311452311323113423110231165231l'
>>> decrypt_text(enc)
'efgh,!@'
```

5 More Realistic Encryption and Decryption

To complete our implementation we need to make our encryption and decryption more bulletproof. One way of doing it is to make our public key longer. The longer the number the harder it is to factorize. For this assignment, let us fix our public key and code separator to the following values. The code separator is `euclid_number(33)`. The `public_key_factors` are the prime factors of `public_key` that you will have to compute.

```
public_key = 614889782588491410
code_separator = 1001464665059919006750923313164994005736633465320043309lL
```

The file `quotes.py` contains several quote by Rumi, Hafez, Khayyam, and Avicenna. We are ready to have some fun encrypting and decrypting them.

```
>>> quote_01
'Raise your words, not voice. It is rain that grows flowers, not thunder.\n\t\t\tJalaluddin Rumi'
>>> enc_01
'5100146466505991900675092331316499400573663346532004330918410014646650599190067509233131649940057
3663346532004330911100146466505991900675092331316499400573663346532004330919810014646650599190067 5
0923313164994005736633465320043309111110014646650599190067509233131649940057366334653200433091116 10
0146466505991900675092331316499400573663346532004330915100146466505991900675092331316499400573663 3
4653200433091117100146466505991900675092331316499400573663346532004330914100146466505991900675092 3
3131649940057366334653200433091102100146466505991900675092331316499400573663346532004330919100146 4
6650599190067509233131649940057366334653200433091381001464665059919006750923313164994005736633465 3
2004330912100146466505991900675092331316499400573663346532004330911221001464665059919006750923313 1
6499400573663346532004330917100146466505991900675092331316499400573663346532004330911141001464665 0
5991900675092331316499400573663346532004330917100146466505991900675092331316499400573663346532004 3
3091120100146466505991900675092331316499400573663346532004330911100146466505991900675092331316499 4
0057366334653200433091115100146466505991900675092331316499400573663346532004330911100146466505991 9
0067509233131649940057366334653200433091331001464665059919006750923313164994005736633465320043309 1
1310014646650599190067509233131649940057366334653200433091137100146466505991900675092331316499400 5
7366334653200433091131001464665059919006750923313164994005736633465320043309112910014646650599190 0
6750923313164994005736633465320043309131001464665059919006750923313164994005736633465320043309111 5
1001464665059919006750923313164994005736633465320043309114100146466505991900675092331316499400573 6
6334653200433091124100146466505991900675092331316499400573663346532004330915100146466505991900675 0
9233131649940057366334653200433091117100146466505991900675092331316499400573663346532004330911110 0
1464665059919006750923313164994005736633465320043309155100146466505991900675092331316499400573663 3
4653200433091410014646650599190067509233131649940057366334653200433091331001464665059919006750923 3
1316499400573663346532004330918100146466505991900675092331316499400573663346532004330911141001464 6
6505991900675092331316499400573663346532004330911100146466505991900675092331316499400573663346532 0
0433091112100146466505991900675092331316499400573663346532004330910100146466505991900675092331316 4
9940057366334653200433091117100146466505991900675092331316499400573663346532004330919100146466505 9
9190067509233131649940057366334653200433091381001464665059919006750923313164994005736633465320043 3
0918100146466505991900675092331316499400573663346532004330911221001464665059919006750923313164994 0
0573663346532004330911100146466505991900675092331316499400573663346532004330911121001464665059919 0
0675092331316499400573663346532004330919100146466505991900675092331316499400573663346532004330911 1
1100146466505991900675092331316499400573663346532004330915100146466505991900675092331316499400573 6
6334653200433091101100146466505991900675092331316499400573663346532004330916100146466505991900675 0
9233131649940057366334653200433091103100146466505991900675092331316499400573663346532004330913100 1
4646650599190067509233131649940057366334653200433091471001464665059919006750923313164994005736633 4
6532004330911410014646650599190067509233131649940057366334653200433091561001464665059919006750923 3
1316499400573663346532004330918100146466505991900675092331316499400573663346532004330917710014646 6
5059919006750923313164994005736633465320043309113100146466505991900675092331316499400573663346532 0
0433091134100146466505991900675092331316499400573663346532004330911210014646650599190067509233131 6
4994005736633465320043309147100146466505991900675092331316499400573663346532004330915100146466505 9
9190067509233131649940057366334653200433091107100146466505991900675092331316499400573663346532004 3
3091610014646650599190067509233131649940057366334653200433091117100146466505991900675092331316499 4
0057366334653200433091111001464665059919006750923313164994005736633465320043309143100146466505991 9
0067509233131649940057366334653200433091010014646650599190067509233131649940057366334653200433091 1
1510014646650599190067509233131649940057366334653200433091121001464665059919006750923313164994005 7
3663346532004330911121001464665059919006750923313164994005736633465320043309121001464665059919006 7
5092331316499400573663346532004330911061001464665059919006750923313164994005736633465320043309121 0
0146466505991900675092331316499400573663346532004330911111001464665059919006750923313164994005736 6
3346532004330911410014646650599190067509233131649940057366334653200433091561001464665059919006750 9
2331316499400573663346532004330911410014646650599190067509233131649940057366334653200433091140100 1
4646650599190067509233131649940057366334653200433091710014646650599190067509233131649940057366334 6
5320043309110710014646650599190067509233131649940057366334653200433091131001464665059919006750923 3
1316499400573663346532004330911151001464665059919006750923313164994005736633465320043309171001464 6
6505991900675092331316499400573663346532004330911191001464665059919006750923313164994005736633465 3
2004330911210014646650599190067509233131649940057366334653200433091471001464665059919006750923313 1
6499400573663346532004330911110014646650599190067509233131649940057366334653200433091114100146466 5
0599190067509233131649940057366334653200433091910014646650599190067509233131649940057366334653200 4
3309112010014646650599190067509233131649940057366334653200433091121001464665059919006750923313164 9
9400573663346532004330911261001464665059919006750923313164994005736633465320043309191001464665059 9
1900675092331316499400573663346532004330911251001464665059919006750923313164994005736633465320043 3
0911010014646650599190067509233131649940057366334653200433091122100146466505991900675092331316499 4
```

```
005736633465320043309131001464665059919006750923313164994005736633465320043309133100146466505991900
675092331316499400573663346532004330910100146466505991900675092331316499400573663346532004330911003
10014646650599190067509233131649940057366334653200433091810014646650599190067509233131649940057366
33465320043309111210014646650599190067509233131649940057366334653200433091101001464665059919006750
92331316499400573663346532004330911181001464665059919006750923313164994005736633465320043309121001
46466505991900675092331316499400573663346532004330911201001464665059919006750923313164994005736633
46532004330916100146466505991900675092331316499400573663346532004330911031001464665059919006750923
31316499400573663346532004330914100146466505991900675092331316499400573663346532004330911151001464
'
```

The encryption of quote_01 has 10,563 digits. A lot more bulletproof than with the previous public key. Let us decrypt it and see what comes out.

```
>>> decrypt_text(enc_01)
'Raise your words, not voice. It is rain that grows flowers, not thunder.\n\t\t\tJalaluddin Rumi'
>>> print(decrypt_text(enc_01))
Raise your words, not voice. It is rain that grows flowers, not thunder.
Jalaluddin Rumi
```

Let us encrypt and decrypt a longer quote.

```
>>> quote_04
'On a day\nwhen the wind is perfect,\nthe sail just needs to open and the world is full of beauty.\n
Today is such a\nday.\n\t\t\tJalaluddin Rumi'
>>> enc_04 = encrypt_text(quote_04)
>>> enc_04
'10100146466505991900675092331316499400573663346532004330918610014646650599190067509233131649940057
36633465320043309114100146466505991900675092331316499400573663346532004330911341001464665059919006
75092331316499400573663346532004330919100146466505991900675092331316499400573663346532004330913810
01464665059919006750923313164994005736633465320043309110100146466505991900675092331316499400573663
34653200433091104100146466505991900675092331316499400573663346532004330913100146466505991900675092
33131649940057366334653200433091331001464665059919006750923313164994005736633465320043309113100146
46650599190067509233131649940057366334653200433091118100146466505991900675092331316499400573663346
53200433091010014646650599190067509233131649940057366334653200433091981001464665059919006750923313
16499400573663346532004330910100146466505991900675092331316499400573663346532004330911221001464665
05991900675092331316499400573663346532004330919100146466505991900675092331316499400573663346532004
33091161001464665059919006750923313164994005736633465320043309161001464665059919006750923313164994
00573663346532004330911211001464665059919006750923313164994005736633465320043309112100146466505991
90067509233131649940057366334653200433091119100146466505991900675092331316499400573663346532004330
91610014646650599190067509233131649940057366334653200433091103100146466505991900675092331316499400
57366334653200433091610014646650599190067509233131649940057366334653200433091112100146466505991900
67509233131649940057366334653200433091410014646650599190067509233131649940057366334653200433091331
00146466505991900675092331316499400573663346532004330911110014646650599190067509233131649940057366
33465320043309112710014646650599190067509233131649940057366334653200433091310014646650599190067509
23313164994005736633465320043309110510014646650599190067509233131649940057366334653200433091110014
64665059919006750923313164994005736633465320043309110210014646650599190067509233131649940057366334
65320043309181001464665059919006750923313164994005736633465320043309136100146466505991900675092331
31649940057366334653200433091510014646650599190067509233131649940057366334653200433091121100146466
50599190067509233131649940057366334653200433091210014646650599190067509233131649940057366334653200
43309110610014646650599190067509233131649940057366334653200433091610014646650599190067509233131649
94005736633465320043309111210014646650599190067509233131649940057366334653200433091141001464665059
91900675092331316499400573663346532004330911241001464665059919006750923313164994005736633465320043
30914100146466505991900675092331316499400573663346532004330913310014646650599190067509233131649940
05736633465320043309151001464665059919006750923313164994005736633465320043309110710014646650599190
06750923313164994005736633465320043309131001464665059919006750923313164994005736633465320043309111
61001464665059919006750923313164994005736633465320043309114100146466505991900675092331316499400573
66334653200433091561001464665059919006750923313164994005736633465320043309181001464665059919006750
92331316499400573663346532004330911161001464665059919006750923313164994005736633465320043309191001
46466505991900675092331316499400573663346532004330911071001464665059919006750923313164994005736633
46532004330914100146466505991900675092331316499400573663346532004330911151001464665059919006750923
31316499400573663346532004330911310014646650599190067509233131649940057366334653200433091120100146
46650599190067509233131649940057366334653200433091610014646650599190067509233131649940057366334653
20043309110310014646650599190067509233131649940057366334653200433091810014646650599190067509233131
64994005736633465320043309110310014646650599190067509233131649940057366334653200433091210014646650
```

```
05991900675092331316499400573663346532004330911171001464665059919006750923313164994005736633465320
04330911010014646650599190067509233131649940057366334653200433091511001464665059919006750923313164
99400573663346532004330914100146466505991900675092331316499400573663346532004330911110014646650599
19006750923313164994005736633465320043309121001464665059919006750923313164994005736633465320043309
11171001464665059919006750923313164994005736633465320043309121001464665059919006750923313164994005
73663346532004330911051001464665059919006750923313164994005736633465320043309121001464665059919006
75092331316499400573663346532004330911021001464665059919006750923313164994005736633465320043309110
10014646650599190067509233131649940057366334653200433091391001464665059919006750923313164994005736
63346532004330915100146466505991900675092331316499400573663346532004330911171001464665059919006750
92331316499400573663346532004330912100146466505991900675092331316499400573663346532004330919810014
64665059919006750923313164994005736633465320043309114100146466505991900675092331316499400573663346
53200433091129100146466505991900675092331316499400573663346532004330912100146466505991900675092331
31649940057366334653200433091109100146466505991900675092331316499400573663346532004330911010014646
65059919006750923313164994005736633465320043309139100146466505991900675092331316499400573663346532
00433091710014646650599190067509233131649940057366334653200433091109100146466505991900675092331316
49940057366334653200433091910014646650599190067509233131649940057366334653200433091123100146466505
99190067509233131649940057366334653200433091121001464665059919006750923313164994005736633465320043
30911301001464665059919006750923313164994005736633465320043309171001464665059919006750923313164994
00573663346532004330911191001464665059919006750923313164994005736633465320043309151001464665059919
00675092331316499400573663346532004330913410014646650599190067509233131649940057366334653200433091
61001464665059919006750923313164994005736633465320043309111210014646650599190067509233131649940057
'
>>> decrypt_text(enc_04)
'On a day\nwhen the wind is perfect,\nthe sail just needs to open and the world is full of beauty.
\nToday is such a\nday.\n\t\t\tJalaluddin Rumi'
>>> print(decrypt_text(enc_04))
On a day
when the wind is perfect,
the sail just needs to open and the world is full of beauty.
Today is such a
day.
			Jalaluddin Rumi
```

This encoding has 15,665 digits.

6 What To Submit

Implement the above functions in Py2 and save them in the file *encrypt.py*. Implement the above functions in Py3 and save them in the file *encrypt_py3.py*. To save you some typing, I have attached the files *encrypt.py* and *encrypt_py3.py* with all the function stubs already defined. You can use those files to implement your solutions.

You will see that the imports at the beginning of Py2 and Py3 files look different. Specifically, imports in *encrypt.py* are as follows:

```
from random import randint
from eucs import *
from fixed_points import half_interval_method
from quotes import *
```

The imports in *encrypt_py3.py* look as follows:

```
from random import randint
from eucs_py3 import *
from fixed_points_py3 import half_interval_method
from quotes import *
```

The files *eucs.py* and *fixed_poins.py* should contain your Py2 implementations of euclid numbers and the half interval method. The files *eucs_py3.py* and *fixed_points_py3.py* should contain your Py3 implementations of the same functionalities. When you debug your programs, place the files in the same directory with *encrypt.py* and *encrypt_py3.py*. Please do not change the signatures and names of the functions in the Py files. It will make the graders' job much easier.

Zip *encrypt.py*, *encrypt_py3.py*, *eucs.py*, *eucs_py3.py*, *fixed_points.py*, and *fixed_points_py3.py* into `hw03.zip` and submit the zip via Canvas.

Happy Hacking!