# CISS451/MATH451: Cryptography and Computer Security
# Assignment 6

This is an assignment on Vigenere cipher.

**Python programming: Characters and mod 26**

For classical ciphers and also many modern applications, you need to convert data from one format to another. For classical ciphers, you need to convert between a-z and mod 26 integers. Modern computer systems have to translate between different data and binary numbers.

To convert from characters to integer ASCII code in Python try this

```
c = 'm'
print ord(c)
```

To convert from ASCII code back to characters try this:

```
print chr(97)
```

Of course if you want `a` to correspond to 0, to convert from character to mod 26 integer, you do this:

```
c = 'm'
print ord(c) - ord('a')
```

and to convert from mod 26 integer to character you do this:

```
print chr(2 + ord('a'))
```

(in 2 mod 26 corresponds to `c`).

So here's a simple function to convert from characters to mod 26:

```
def to_int(c):
    return ord(c) - ord('a')
```

It's a good idea to check that c is a string of length 1. (Python doese not have the concept of characters, but only of strings. A character is just a string of length 1.) To enforce that c is a string of length 1 we do this:

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not a..z" % c)
    return ord(c) - ord('a')
```

You don't have to know how those two checks work. You can just use the code above. To find out more about Python you can take the Python programming class. Here's the code to convert from integer mod 26 to character:

```
def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s in on 0..25" % i)
    return chr(i + ord('a'))
```

Altogether we have these functions to convert between a-z and mod 26:

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s in on 0..25" % i)
    return chr(i + ord('a'))
```

It's also a good idea to have a similar function to convert a string (i.e. length not necessarily 1) to a list of integers. So we do this:

```
def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints
```

```
def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs
```

Altogether we have

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s in on 0..25" % i)
    return chr(i + ord('a'))

def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints

def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs

print "test ..."
print to_chrs([0,1,2,3,4,5])
print to_ints(['a', 'c', 'e', 'z'])
```

Make sure you study and run the above code.

Note that in the above, for the `to_chrs`, you get a list of characters. Sometimes it's more readable to print a string rather than the characters. So it might be a good idea to know how to convert between strings and lists of characters. You can easy get list of characters from a string:

```
s = "jump"
cs = list(s)
print cs
```

And from characters to string you do this:

```
cs = ['j', 'u', 'm', 'p']
s = ''.join(cs)
print s
```

**Python Programming: Cleaning up a string**

For classical ciphers, you usually convert uppercase to lowercase and throw away the non a-z. That's easy to do in Python. First in Python run

```
import string
print string.ascii_lowercase
print 'A' in string.ascii_lowercase
print '!' in string.ascii_lowercase
print 'b' in string.ascii_lowercase
```

```
import string
def cleanup(cs):
    cs = cs.lower()
    xs = []
    for c in cs:
        if c in string.ascii_lowercase:
            xs.append(c)
    return ''.join(xs)

print cleanup("Hello, world!")
```

## Python Programming: Pretty print

To the above code, we add a simple function to print messages with when given a window size. This might help when for instance when working in a group and you need to specify a the location of a character. The function assumes that the string is made up of lowercase characters.

```
def printstr(s, columns=50, omit_int=False):
    import sys
    print "     ",
    for i in range(columns):
        if i % 10 == 0:
            sys.stdout.write("%s" % (i/10))
        else:
            sys.stdout.write(" ")
    if not omit_int:
        sys.stdout.write(" ")
        for i in range(columns):
            if i % 10 == 0:
                sys.stdout.write("%3s" % (i/10))
            else:
                sys.stdout.write("   ")
    print
    print "     ",
    for i in range(0, columns):
        sys.stdout.write("%s" % (i % 10))
    if not omit_int:
        sys.stdout.write(" ")
        for i in range(0, columns):
            sys.stdout.write("%3s" % (i % 10))
    print
    def drawline():
        sys.stdout.write("    +" + "-" * columns + "+")
        if not omit_int:
            sys.stdout.write("---" * columns + "+")
        print
    drawline()
    line = 0
    while s != "":
        t = s[:columns]
        t = t + " " * (columns - len(t)) + "|"
        sys.stdout.write("%4s|%s" % (line, t))
        if not omit_int:
            t = s[:columns]
            t = to_ints(t)
            sys.stdout.write("".join(["%3s" % _ for _ in t]))
            sys.stdout.write("   " * (columns - len(t)))
            sys.stdout.write("|")
        print
        s = s[columns:]
        line += 1
    drawline()

print "test printstr ..."
s = """
Dear reader! it rests with you and me whether, in our two fields of action
similar things shall be or not. Let them be!
We shall sit with lighter bosoms on the hearth, to see the ashes
 of our fires turn grey and cold.
"""
s = cleanup(s)
printstr(s, 30, False)
```

The output of the test is

```
     0         1         2         3         4         5
     012345678901234567890123456789012345678901234567890123456789
    +------------------------------------------------------------
   0|dearreaderitrestswithyouandmewhetherinourtwofieldsofactionsi
   1|milarthingsshallbeornotletthembeweshallsitwithlighterbosomso
   2|nthehearthtoseetheashesofourfiresturngreyandcold
```

Here's another execution with the last argument set to True:

```
>>> printstr(s, 20, False)
     0         1           0                             1
     01234567890123456789   0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9
    +--------------------+------------------------------------------------------------+
   0|dearreaderitrestswit|  3  4  0 17 17  4  0  3  4 17  8 19 17  4 18 19 18 22  8 19|
   1|hyouandmewhetherinou|  7 24 14 20  0 13  3 12  4 22  7  4 19  7  4 17  8 13 14 20|
   2|rtwofieldsofactionsi| 17 19 22 14  5  8  4 11  3 18 14  5  0  2 19  8 14 13 18  8|
   3|milarthingsshallbeor| 12  8 11  0 17 19  7  8 13  6 18 18  7  0 11 11  1  4 14 17|
   4|notletthembeweshalls| 13 14 19 11  4 19 19  7  4 12  1  4 22  4 18  7  0 11 11 18|
   5|itwithlighterbosomso|  8 19 22  8 19  7 11  8  6  7 19  4 17  1 14 18 14 12 18 14|
   6|nthehearthtoseetheas| 13 19  7  4  7  4  0 17 19  7 19 14 18  4  4 19  7  4  0 18|
   7|hesofourfiresturngre|  7  4 18 14  5 14 20 17  5  8 17  4 18 19 20 17 13  6 17  4|
   8|yandcold            | 24  0 13  3  2 14 11  3                                    |
    +--------------------+------------------------------------------------------------+
```

**Python Programming: The code**

```
import string, sys

def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s in on 0..25" % i)
    return chr(i + ord('a'))

def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints

def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs

def cleanup(cs):
    cs = cs.lower()
    xs = []
    for c in cs:
        if c in string.ascii_lowercase:
            xs.append(c)
    return ''.join(xs)

def printstr(s, columns=50, omit_int=False):
    print "     ",
    for i in range(columns):
        if i % 10 == 0:
            sys.stdout.write("%s" % (i/10))
        else:
            sys.stdout.write(" ")
    if not omit_int:
        sys.stdout.write(" ")
        for i in range(columns):
            if i % 10 == 0:
                sys.stdout.write("%3s" % (i/10))
            else:
                sys.stdout.write("   ")
    print
    print "     ",
    for i in range(0, columns):
        sys.stdout.write("%s" % (i % 10))
    if not omit_int:
        sys.stdout.write(" ")
        for i in range(0, columns):
            sys.stdout.write("%3s" % (i % 10))
    print
    def drawline():
        sys.stdout.write("    +" + "-" * columns + "+")
        if not omit_int:
            sys.stdout.write("---" * columns + "+")
        print
    drawline()
    line = 0
    while s != "":
        t = s[:columns]
        t = t + " " * (columns - len(t)) + "|"
        sys.stdout.write("%4s|%s" % (line, t))
        if not omit_int:
            t = s[:columns]
            t = to_ints(t)
            sys.stdout.write("".join(["%3s" % _ for _ in t]))
            sys.stdout.write("   " * (columns - len(t)))
            sys.stdout.write("|")
        print
        s = s[columns:]
        line += 1
    drawline()

print "test to_chrs and to_ints..."
print to_chrs([0,1,2,3,4,5])
print to_ints(['a', 'c', 'e', 'z'])
print "test cleanup ..."
print cleanup("Hello, world!")

print "test printstr ..."
s = """
Dear reader! it rests with you and me whether, in our two fields of action
similar things shall be or not. Let them be!
We shall sit with lighter bosoms on the hearth, to see the ashes
 of our fires turn grey and cold.
"""
s = cleanup(s)

printstr(s, 30, True)
printstr(s, 20, False)
```

Make sure you underestand the above code and try out the test code to underestand how to use the functions.

By the way, in the above, we're not using any long integers in Python. So the above can be easily translated into for instance C++ if you wish.

## Python Programming: table lookup

In C++ you have the concept of arrays. If you have an array of size 10. An array is like a table lookup. For instance support `x` is an array of size 5 with the following values

```
+-------+-------+
| index | value |
+-------+-------+
|     0 |   100 |
|     1 |   -50 |
|     2 |    26 |
|     3 |   990 |
|     4 |    -1 |
+-------+-------+
```

You have already seen that Python has lists which are lilke arrays. There's actually another table lookup called the dictionary. Each entry in a dictionary is key-value pair or a row. The difference between dictionary and lists is the you access row using a key value instead of an index value. Run the following in Python:

```
freq = {} # empty dictionary
freq['a'] = 0
freq['b'] = 0
print freq

freq['a'] = freq['a'] + 1
freq['b'] = freq['b'] + 23
print freq
```

Here's how you think of the `freq` dictionary:

```
+-----+-------+
| key | value |
+-----+-------+
| 'a' |     1 |
| 'b' |    23 |
+-----+-------+
```

You can iterate through a dictionary if you have all the keys:

```
freq = {} # empty dictionary
freq['a'] = 0
freq['b'] = 0
print freq

freq['a'] = freq['a'] + 1
freq['b'] = freq['b'] + 23
print freq

for key in ['a', 'b']:
    print freq[key]
```

You can also obtain the list of keys from the dictionary:

```
freq = {} # empty dictionary
freq['a'] = 0
freq['b'] = 0
print freq

freq['a'] = freq['a'] + 1
freq['b'] = freq['b'] + 23
print freq

for key in ['a', 'b']:
    print freq[key]

for key in freq.keys():
    print freq[key]
```

If you access a row using a non-existent key you get an error. For instance the last statement will get you into trouble:

```
freq = {} # empty dictionary
freq['a'] = 0
freq['b'] = 0
print freq

freq['a'] = freq['a'] + 1
```

```
freq['b'] = freq['b'] + 23
print freq

for key in ['a', 'b']:
    print freq[key]

for key in freq.keys():
    print freq[key]

print freq['c']
```

Instead of getting an error, you can also lookup a dictionary and when the key is not, request for a default value. For instance:

```
freq = {} # empty dictionary
freq['a'] = 0
freq['b'] = 0
print freq

freq['a'] = freq['a'] + 1
freq['b'] = freq['b'] + 23
print freq

for key in ['a', 'b']:
    print freq[key]

for key in freq.keys():
    print freq[key]

print freq.get('c', 0) # if 'c' is not a key, gimme 0
```

Dictionaries can be used for frequency analysis in ciphers. For instance this creates a frequency table:

```
s = "helloworld"

freq = {}
for c in s:
    print "processing", c
    freq[c] = freq.get(c, 0) + 1

import string
print "char freq analysis of s ..."
for c in string.ascii_lowercase:
    print c, freq.get(c, 0)
```

Here's an analysis of 2–grams

```
s = "tobeornottobethatisthequestion"

freq = {}
for i in range(len(s) - 1):
    key = s[i:i+2]
    print "processing", key
    freq[key] = freq.get(key, 0) + 1

import string
print "2-gram freq analysis of s ..."
for key in freq.keys():
    print key, freq[key]
```

The syntax

```
s[i:i+2]
```

returns a substring from string `s` beginning at index `i` and ending at `i+2`, i.e., this is a substring of length 2 starting at index `i`. You can also do this with lists. Here are some examples for you to try out:

```
xs = [11,22,33,44,55,66,77,88,99]
ys = xs[3:8]
print ys

xs = "hello world"
ys = xs[3:8]
```

You can also specifiy the amount of step:

```
xs = [11,22,33,44,55,66,77,88,99]
ys = xs[3:10:2] # every other character from index 3 to 10
print ys

xs = "hello world"
ys = xs[3:10:2]
```

This will help in cutting up strings/lists into pieces when you do Vigenere.

**Q1.** You are given the following message $m$:

*Dear reader! It rests with you and me whether, in our two fields of action similar things shall be or not. Let them be! We shall sit with lighter bosoms on the hearth, to see the ashes of our fires turn grey and cold.*

Perform the following:

1. Clean it up (i.e. change uppercase to lowercase, remove all characters which are not `a-z`.
2. Change the message from (a) to mod 26 integers.
3. Encrypt the message from (b) using Vigenere with the key of

   ```
   times
   ```

4. Convert the message from mod 26 to lowercase characters.
5. Print the ciphertext from the above using the `printstr` function with `columns` parameter set to 50 and last parameter set to `True`, i.e. do not print the mod 26 integers. In other words if the ciphertext as a string is `s`, execute `printstr(s, 50, True)` and paste the output below in `q1.tex`.

**SOLUTION.**

```
   0         1         2         3         4
   01234567890123456789012345678901234567890123456789
  +--------------------------------------------------+
 0|bkwhlcgzulgznumrysynfekkuljiuqfkpxypojeopzsezgkhtm|
 1|mlwsngujickohqlrnedaqydqfjhaellupbyrzdugzksumfghbm|
 2|gzsynfrewbrknriquiiilzdubcgnjbruouyrnaqmfkoez     |
  +--------------------------------------------------+
```

Q2. The goal is to find the key used for the cipher that produced the following ciphertext and to decrypt the message.

```
mpqzalqfsjaihmfzafvgetqhlhbtiobvpsotvpfwbvsxzxvqry
toqhagtascbvsgskmxikltksmmemwslczqgompfqmpuwafxdik
lqhiwgbdcslumruhcxhhhaemtegnizxafsgwetmkmturymwtme
lmxjobbtedeqyeybvmfdxkasdgmewobbtlalpmxkmqxpggizhs
vmdxsbvmmjhnqbztcexahvgtggpuqagxmvltzuwagorvgfmjgw
lauzwlcyqwkizhagxmvlyzaqwqkqwkbdqkwgbupamgrsjbbiek
mwnikxmzaamptedyiziqxbtelamiektbtsjhcslyxvfpwfizqs
wmfslamysvxtajlamfmexeqejrwrinxzkxzbvsefwxgxlbvsrg
fwdixtqflagizclaqzklaizpmvqrij
```

(a) For $m = 1$, compute the $I$–value for the above string.

(b) For $m = 2$, compute the $I$–values for the two strings. `mqa...` and `pzl....` Compute the average $I$–value.

(c) For $m = 3$, compute the $I$–values for the three strings. Compute the average $I$–value.

(d) Do the same for $m = 4$.

(e) Do the same for $m = 5$.

(f) Do the same for $m = 6$.

(g) What is the most likely keylength $m$?

(h) Compute and list the top $m - 1$ $M$–values.

(i) Write down $m - 1$ mod 26 equations from (h)

(j) Express the mod 26 equations from (i) in terms of $k_0$.

(k) For $k_0 = 0, 1, 2, ..., 25$, write down all the keys (using lowercase `a-z`.)

(l) What is the key from (k)? (State this as a string in lowercase and not as a sequence of mod 26 integers.)

(j) Decrypt the message.

**SOLUTION.** thevisitorhavingstrolledtothewindowandbeingthenengagedinlookingcarelesslyou twasasunmovedbythisimpressiveentryasmancouldpossiblybehestoodwhistlingto himselfwith-

allimaginablecoolnesswithhishatstillonandacertainairofexhaustion uponhiminpartarisingfromex-
cessivesummerandinpartfromexcessivegentility foritwastobeseenwithhalfaneyethathewasathor-
oughgentlemanmadeto themodelofthetimewearyofeverythingandputtingnomorefaithinanythingthanlucif

M is 5 this is the key length
key is [19, 8, 12, 4, 18] ”times”

”””the visitor having strolled to the window and being then engaged in looking carelessly out was as unmoved by this impressive entry as man could possibly be he stood whistling to himself with all imaginable coolness with his hat still on and a certain air of exhaustion upon him inpart arising from excessive summer and in part from excessive gentility for it was to be seen with half an eye that he was a thorough gentleman made to the model of the time weary of everything and putting no more faith in anything than lucifer”””