Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2018-2)

Tarea 00

Entrega

- **Tarea**
  - **Fecha y hora:** 26 de agosto de 2018, 23:59.
  - **Lugar:** GitHub — Carpeta: `Tareas/T00/`
- **README.md**
  - **Fecha y hora:** 27 de agosto de 2018, 23:59.
  - **Lugar:** GitHub — Carpeta: `Tareas/T00/`

Objetivos

- Desarrollar algoritmos para la resolución de problemas complejos.
- Aplicar competencias asimiladas en *Introducción a la programación* para el desarrollo de una solución a un problema.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Trabajar con archivos de texto para leer, escribir y procesar datos.
- Escribir código utilizando paquetes externos (*i.e.* código no escrito por el estudiante), como por ejemplo, módulos que pertenecen a la librería estándar de Python.

Índice

1. Introducción

¡Suenen las alarmas! ¡Ha ocurrido un ataque cibernético en el correo electrónico del departamento de ayudantes de *Programación avanzada*! El causante no es ni más ni menos que el malvado Dr. H^4, por lo que, con sus maliciosas habilidades de inyección de código (sí, eso realmente existe, para aprender más de esto haz clic **aquí**) ha logrado destruir toda la infraestructura de los correos y sólo ha dejado unas pequeñas bases de datos atrás para reírse ~~malvadamente~~ desde la ~~in~~comodidad de su guarida. Es por esto que se te ha encargado la *única* y *fabulosa* tarea de rehacer este sistema de correos electrónicos para poder facilitar que los ayudantes hagan bien su trabajo.

2. *DCCorreos*

En esta tarea, se tendrá que crear el programa *DCCorreos*, un sistema de manejo de correos electrónicos tal como *Gmail* y *Outlook* en conjunto con un sistema de manejo y administración de calendarios y eventos. Con esto, un usuario va a poder crear una dirección de correo y podrá enviar correos electrónicos a otros, además de crear eventos en su calendario personal e invitar gente a estos. Para esto, se debe utilizar los conocimientos adquiridos en *Introducción a la programación*, donde corresponda, para modelar la siguiente tarea y mediante el uso de la consola, permitir la interacción del usuario.

3. Funcionalidades del programa

3.1. Registro e inicio de sesión

Al igual que un servicio de correo electrónico normal, *DCCorreos* debe tener un menú de inicio en el cual la persona que se encuentre en el computador pueda acceder a su cuenta, o crear una nueva cuenta. El correo registrado debe ser único y debe tener una contraseña. Esto implica tener un menú de inicio en el cual se pueda acceder al correo pidiendo el correo electrónico y su contraseña asociada, verificando que la contraseña sea correcta al momento de entrar, llevando a los menús correspondientes. Por otra parte, si se decide crear una cuenta se debe pedir la dirección de correo electrónico que se quiera utilizar y la contraseña que se desea tener.

Para asegurar la consistencia y unicidad de cada correo electrónico, el sistema solo va a permitir que existan dos usuarios con igual nombre de correo, siempre y cuando el servicio que provee ese correo sea diferente. Por ejemplo si se acepta lo siguiente:

`h4@uc.cl` y `h4@ing.puc.cl`

Pero **no** se acepta tener dos usuarios con el mismo nombre **y** mismo proveedor de servicio web.

Al momento que un usuario decida registrarse se le preguntará por un nombre y una contraseña con un largo mayor o igual a seis caracteres. Una vez iniciado sesión, el usuario podrá disfrutar de todas las funcionalidades del servicio electrónico de *DCCorreos*. Cabe notar que siempre se debe tener "@" y una dirección de servicio a continuación, es decir, no se puede tener un correo sin el "@dirección.de_servicio". Además, la contraseña debe tener como mínimo 6 caracteres y ni la contraseña ni el nombre de usuario deben tener "," o ";".

3.2. Correos

Entre las opciones existentes en tu programa deberá existir la opción "Enviar un correo". Al seleccionar esta opción tu programa deberá seguir la siguiente secuencia:

1. **Seleccionar destinatarios**: En este paso el usuario deberá escribir los correos electrónicos de los usuarios a los cuales se les quiere enviar su mensaje, tomando en cuenta que un correo puede ser enviado a una o más personas. Si hay más de un destinatario, las direcciones de correo deben ser escritas separadas por comas. Por ejemplo, `jecastro1@uc.cl,fidelrio@uc.cl,cruz@uc.cl`
2. **Asunto del correo**: Cada correo debe tener un asunto que no puede ser vacío. Esta palabra o oración será lo único que podrá ver el usuario antes de abrir el correo en su bandeja, es por esto que el asunto no debe sobrepasar los 50 caracteres.
3. **Cuerpo del correo**: En este paso el usuario podrá escribir lo que sea: sus reclamos a Banner, pedir vacantes a la DiPre, etcétera. Lo único que interesa es que el usuario pueda enviar un mensaje **siempre y cuando no exceda** los 256 caracteres, condición que se debe revisar antes de enviar el correo.
4. **Opciones del correo**: De forma opcional el usuario puede marcar (una o varias) clasificaciones del correo. Las clasificaciones pueden ser: "`Importante`", "`Publicidad`", "`Destacado`" y "`Newsletter`". Si el correo no tuviese clasificación entonces, por defecto, será "`sin clasificación`".
5. **Enviar**: El usuario finalmente podrá enviar el correo electrónico.

3.3. Bandeja de entrada

Al igual que en un servicio clásico de *emails*, una cuenta de correo debe tener una bandeja de entrada para ver todos los correos recibidos[1]. Para esto, se deben mostrar todos los correos recibidos y poder acceder a ellos mediante la selección del correo deseado. El usuario deberá poder ver en su bandeja todos los correos recibidos ordenados desde el más reciente al más antiguo. En la bandeja sólo de debe mostrar el tipo de correo (que se configura en las opciones del correo al momento de enviarlo) y el asunto. A modo de ilustración, el siguiente ejemplo puede servir como una bandeja de entrada (no necesariamente debe ser igual).

```
Bandeja de Entrada: cilopez

  1.- Publicidad          | Portate a Clavistel
  2.- Importante          | Enunciado T01 IIC2133
  3.- sin clasificación   | Paga la cuota del asado
  4.- Destacado           | Vidal transferido al Barcelona
```

Finalmente, si el usuario desea revisar un correo específico, este al seleccionarlo deberá desplegar toda la información en la consola.

3.4. Calendario

Además de los correos, cada usuario tiene acceso a un calendario en el que puede consultar, agregar, editar, remover e invitar a otros usuarios a los eventos. En todo momento cada usuario sólo puede ver los eventos que creó o a los que fue invitado.

[1] De esta manera, el cuerpo de ayudantes ~~no pierde las recorrecciones de las tareas~~ puede funcionar en forma correcta.

3.4.1. Eventos

Cada evento debe tener:

1. **Propietario**: Dirección de correo electrónico **completa** del usuario que creó el evento.
2. **Nombre**: Nombre del evento, debe contener como mínimo 6 caracteres y como máximo 50.
3. **Fecha y hora de inicio y término**: Dos fechas y horas en que el evento se lleva a cabo, una para el inicio y otra para el término del evento. Ejemplo: (2018-08-07 14:00:00, 2018-08-07 15:30:00). Como es esperable, la fecha de término no puede ser anterior a la de inicio. El usuario puede no dar una hora de término, en tal caso la hora de término sera una hora después del inicio. La librería `datetime`, incluida en Python, te puede ser de gran utilidad para manejar las fechas.
4. **Descripción**: Texto con detalles sobre el evento, en caso que el usuario no quiera dar una descripción entonces por defecto se debe guardar como descripción: "`sin descripcion`".
5. **Invitados**: Un listado de las direcciones de correo de los usuarios invitados. Si no hay invitados, entonces por defecto se tendrá "`sin invitados`"
6. **Etiquetas**: Un listado de etiquetas que ayuden a filtrar los eventos. Las etiquetas son *strings*. Si un evento no tiene etiquetas, entonces este por defecto estará etiquetado como "`sin etiquetas`".

Además, no pueden haber dos eventos con la mismas fechas y el mismo nombre. Para asegurar unicidad de los datos, se sugiere trabajar con un identificador único para cada evento.

3.4.2. Acciones disponibles

Al acceder al calendario se debe poder **crear un nuevo evento**. Para crear un nuevo evento se le debe pedir al usuario toda la información que el evento debe contener (ver 3.4.1). Si se quiere ingresar más de un invitado o etiqueta, se deben escribir las diferentes direcciones de correo o etiquetas separadas por comas (de la misma forma que con los destinatarios de correos).

También se deben dar las opciones para consultar eventos según fecha, nombre y etiquetas. Como resultado se debe mostrar una lista con los nombres de los eventos, y para cada evento, la opción de ver toda su información o ejecutar las siguientes operaciones:

- **Editar evento**: Se deben dar opciones para editar cualquiera de las componentes de un evento, a excepción del propietario que no puede cambiar. Un usuario solo puede editar eventos que él haya creado.
- **Eliminar evento**: Así como su nombre lo dice, esta opción elimina el evento seleccionado. Un usuario sólo puede borrar eventos que él haya creado.
- **Invitar a otros usuarios**: Si el usuario quiere invitar a más gente de la que ya invitó al crear el evento, lo puede hacer en esta opción. Para invitar a otro usuario debe ingresar la dirección de correo del usuario a invitar. Si quiere invitar a más de un usuario, entonces debe ingresar las direcciones de correo separadas por una coma. Sólo el propietario de un evento puede invitar a otros usuarios.

3.5. Encriptación

Una parte importante de los correos electrónicos actuales es que estos estén encriptados para evitar que ciberbandidos puedan interceptar los mensajes y leerlos, o al menos hacer más difícil esta labor. Es por esto que se debe implementar un **"Cifrado César"**. Este cifrado consiste en cambiar el *string* recibido

como *input* y cifrarlo desplazando los caracteres en un cierto número. Por ejemplo: el "Cifrado César" con un desplazamiento de `-10` en cada carácter ASCII[23] "`ryvk`" quedaría:

`ryvk` se tornaría en: `hola`

Para efectos de esta tarea, se pedirá **sumar 10** al orden `ASCII` de cada carácter. Adicionalmente, siendo gente del DCC, los ayudantes aprovecharon la idea de mejorar su seguridad, dándole una capa adicional de encriptación con el método *CipherSaber*. Una versión simplificada de este protocolo funciona de la siguiente manera:

1. Los puntos que se van a comunicar se ponen de acuerdo en una clave de largo máximo 246 caracteres. El tamaño del mensaje a ser enviado no puede exceder los 256 caracteres. Para casos de esta tarea la clave va a ser 2233.

2. Luego se transforma el mensaje a enviar en una cadena de ceros y unos siguiendo estos pasos:

   a) Cada carácter del mensaje es transformado en un número entero positivo según la codificación ASCII. Por ejemplo, los números de esta codificación para la cadena `hola` son 104, 111, 108 y 97.

   b) Luego, el número ASCII es transformado a la correspondiente secuencia binaria de 8 dígitos. Si ésta tiene menos de 8 dígitos, entonces se agregarán ceros a la izquierda hasta obtener una cadena de ocho dígitos.

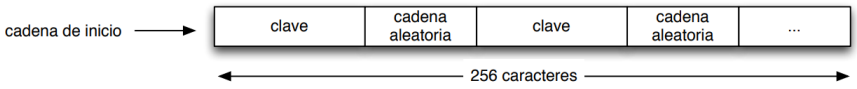
   c) Finalmente todas las representaciones binarias son unidas en una sola cadena de texto.

   La siguiente tabla muestra la codificación en ceros y unos del mensaje "hola"

Carácter	Código ASCII	Codificación binaria (1's y 0's)
h	104	01101000
o	111	01101111
l	118	01101100
a	97	01100001
hola	104 111 118 97	01101000011011110110110001100001

3. Luego, se crea la cadena aleatoria que consiste de una secuencia de diez dígitos elegidos al azar entre 0 y 9. Para descifrar el mensaje esta cadena aleatoria es obtenida de los primeros 10 elementos del mensaje a descifrar.

4. Se crea la cadena de inicio uniendo la clave con la cadena aleatoria (obtenida del paso 3). Esto se repite hasta obtener una secuencia de 256 caracteres. Hay que tener en cuenta que no siempre se tendrá que repetir todo para obtener exactamente la cantidad de caracteres requeridos, para esto solo se tendrá que copiar la parte necesaria.

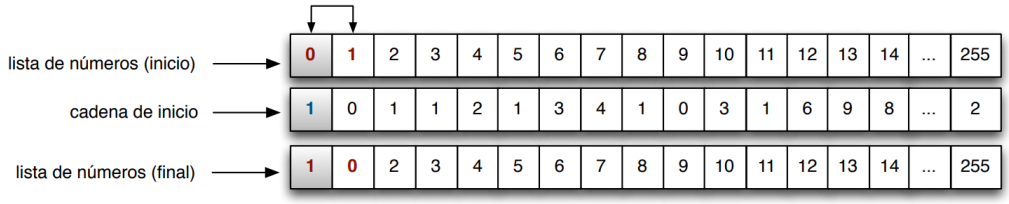


5. Hay que crear una lista de los números 0 hasta 255, luego cambiar el orden de todos los números de esta lista de la siguiente forma: el número de la posición i se intercambia con el número de la posición j, siendo j el valor de la suma de i y el valor obtenido de la cadena de inicio en el índice i. Por ejemplo:

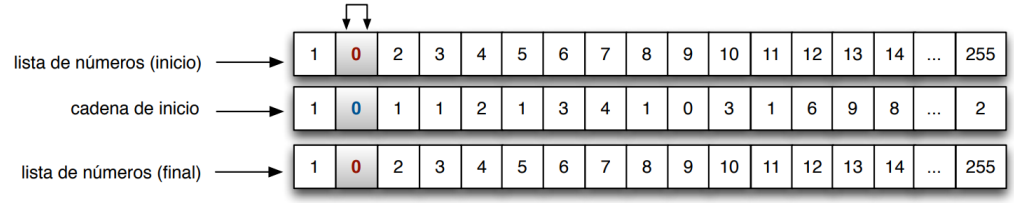
[2] **Link a tabla de caracteres ASCII**.

[3] *Hint:* existe una función en Python que, dado un carácter, entrega el orden ASCII y otra que dado un número entrega ese número en binario.

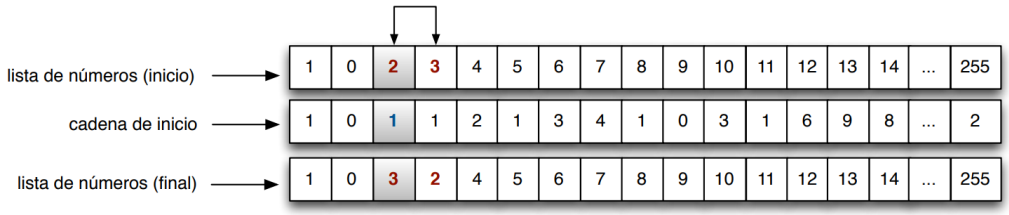
a) El número de la posición "0" (i= 0) será intercambiado con el número de la posición $j = 0 +$ cadena de inicio[0] $= 0 + 1 = 1$.



b) Luego, el número de la posición 1 ($i = 1$), será intercambiado con el de la posición $j = 1+$ cadena de inicio[1] $= 1 + 0 = 1$ (que es lo mismo que no sea intercambiado).



c) De la misma forma, el número de la posición 2 ($i = 2$), será intercambiado con el de la posición $j = 2 +$ cadena de inicio[2] $= 2 + 1 = 3$. Esto se repite para todos los elementos de la lista.



d) El último número de la lista (posición 255) es un caso especial ya que eventualmente será intercambiado con el siguiente, y este no existe. Para este caso considera que el siguiente elemento al último de la lista es el de la posición 0, el subsiguiente el de la posición 1, y así sucesivamente.

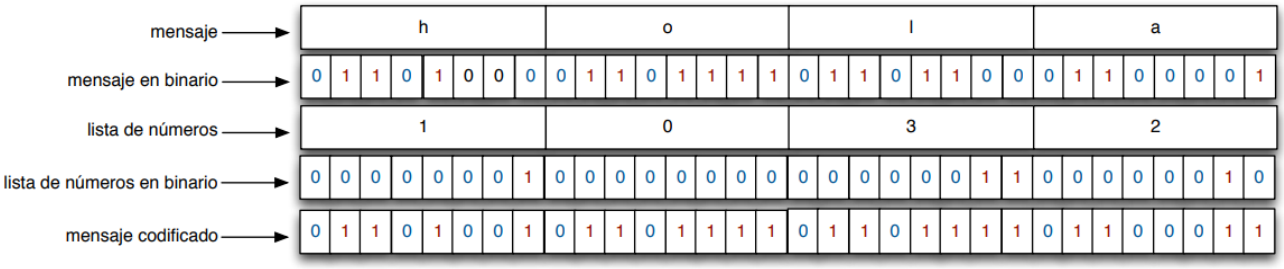
6. Finalmente se codifica el mensaje siguiendo los siguientes pasos:

a) Cada carácter del mensaje es transformado a un secuencia binaria de ocho dígitos de acuerdo al paso 2. Luego se crea una cadena concatenando todas estas secuencias binarias.

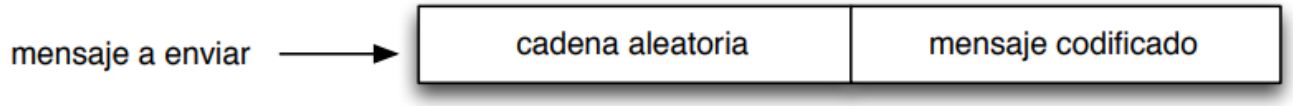
b) Cada elemento de la lista de números obtenida en el paso 5 es transformado a una secuencia binaria de ocho dígitos de acuerdo al paso 2. Luego, se genera una cadena concatenando todas estas secuencias.

c) Con las secuencias binarias de los pasos 6a y 6b, se genera finalmente el mensaje codificado. Este mensaje consiste en una cadena de dígitos binarios que cumplen lo siguiente: el dígito en la posición i del mensaje codificado es un 0 si el dígito en la posición i de la cadena obtenida en el paso 6a es igual al dígito en la posición i de la cadena obtenida en el paso 6b. De lo contrario, el dígito en la posición i del mensaje codificado será un 1...

Por ejemplo, si el mensaje a enviar es "hola" y la lista de números modificada empieza en [1, 0, 3, 2], el mensaje codificado resultante es tal como se muestra en la siguiente figura.



7. Finalmente, el mensaje a enviar es la unión de la cadena aleatoria del paso 3 y el mensaje codificado del paso 6.



Es claro ver que el mensaje codificado no tiene ningún significado para alguien que no conoce la clave, por lo que si alguien intercepta lo que se envió, no puede obtener mucha información sobre el mensaje original.

Evidentemente, el envío debe ser codificado mediante ambas formas. Primero aplicando el "Cifrado César" y luego aplicando sobre este mensaje ya cifrado el "*CipherSaber*" para guardar el mensaje en un archivo `.csv` que registre el mensaje **codificado**. Esta debe tener el mismo formato que se entrega en la base de datos `db_emails.csv` que se describe en el punto 2 de la sección 4.1, en donde cada línea del archivo representa un correo, y está en el formato `from,to,title,body,classifications`, con la primera línea siendo `from,to,title,body,classifications` textualmente.

Además, debes el mensaje codificado al otro correo, para que así cuando se abra se lea el archivo que se creó y se decodifique. Para esto último se debe aplicar el proceso inverso de encriptación "*CipherSaber*" y luego a este mensaje aplicarle el "Cifrado César" inverso (en vez de sumarle 10 a todos los caracteres se les restará 10), con lo que finalmente se obtendrá el mensaje enviado por el usuario y el destinatario así podrá leerlo.

Lo único que se debe encriptar es el mensaje que lleva el correo electrónico, es decir, el cuerpo. El resto del correo no es necesario de encriptar.

4. Base de datos

Como te habrás dado cuenta, existen datos de *DCCorreos* que deben ser persistentes en el tiempo, como por ejemplo los usuarios y sus contraseñas. Para poder mantener esta información es necesario que almacenes en tus propias bases de datos. Estas pueden consistir en archivos `.txt` o `.csv` con un formato que te acomode a ti.

Este tipo de archivos se trabaja igual que un archivo en formato `.txt`, por lo que es meramente notación de lo que se encuentra dentro del archivo. Para todos estos archivos la primera línea es lo que se conoce como el *header* que describe lo que contiene cada columna.

Los datos que deben ser respaldados son:

- Usuarios
- Correos enviados
- Correos guardados en papelera
- Eventos

Esta información debe estar respaldada en una carpeta llamada `datos`. Para que se escriba la información correctamente, es necesario que cuando almacenes información utilices *paths* relativos. Para que entiendas esto mejor te mostraremos un pequeño ejemplo en donde se usan *paths* absolutos y relativos.

```
def path_absoluto():
    with open('C:\Users\Camilo\Desktop\Progra\T00\datos\archivo.txt', 'w') as archivo:
        archivo.write("Esto es un path absoluto")
```

En este caso no existirían problemas en escribir el archivo si tu usuario de Windows se llama Camilo y tienes la tarea en una carpeta llamada "Progra". Sin embargo, probablemente el ayudante no se llame así o la carpeta no tenga el mismo nombre, lo que haría que tu programa sí funcione en tu computador pero no pueda ser corregido correctamente. Frente a esta problemática es necesario usar *paths* relativos y no depender de los nombres de las carpetas.

```
def path_relativo():
    # Para buscar la carpeta 'datos', el intérprete mira
    # dentro de la carpeta en que se ejecuta el programa.
    path = 'datos/archivo.txt'
    with open(path, 'w') as archivo:
        archivo.write("Esto es un path relativo")
```

De esta manera podrás guardar y acceder a tus datos de manera más general.

4.1. Archivos entregados

Afortunadamente, el malvado Dr. H[4] no logró borrar todos los datos que existían en las bases de datos de *DCCorreos*. Los archivos que se salvaron fueron los siguientes:

1. `db_users.csv`: Este archivo contiene los correos electrónicos de los usuarios que se lograron salvar del ataque junto con sus respectivas contraseñas. Hay un usuario por cada fila, donde cada fila es de la forma `user,password`.

Nombre	Tipo de dato	Comentarios
`user`	`string`	Dirección de correo de cada usuario.
`password`	`string`	Contraseña de cada usuario.

Cuadro 1: Estructura `db_users.csv`

2. `db_emails.csv`: Este archivo contiene un registro de los correos electrónicos que ha sido enviados, y por lo tanto recibidos por las diferentes personas a las que les fue enviado el correo. Adicionalmente, también contiene el título del correo y el cuerpo **no encriptado**. Estos dos campos vienen enmarcados con comillas, para poder distinguir entre una "," que es parte del texto y una "," que es

la separadora de los valores. Cada destinatario está separado con un ";" para así evitar confusiones al igual que las clasificaciones que el correo contiene. Cada línea del archivo representa un correo, y está en el formato `from,to,title,body,classifications`.

Nombre	Tipo de dato	Comentarios
`from`	`string`	Dirección de correo de quien envía el correo.
`to`	`lista de strings`	Direcciones de correo de los destinatarios, separadas con ";".
`title`	`string`	Título del correo, entre comillas simples.
`body`	`string`	Cuerpo o contenido del correo, entre comillas simples.
`classifications`	`lista de strings`	Lista de clasificaciones, separadas con ";".

Cuadro 2: Estructura `db_emails.csv`

3. `db_events.csv`: Este archivo contiene un registro de eventos que han ocurrido y algunos que aún no. Además, cada evento viene junto con su nombre, sus respectivas fechas de inicio y término, su descripción, los usuarios que están invitados y el dueño del evento. Al igual que en la base de datos de los correos, el nombre y la descripción del evento se encuentran entre comillas, mientras que la lista de invitados están separados por un ";". El archivo tiene el formato que está a continuación para cada evento: `owner,name,start,finish,description,invited`.

Nombre	Tipo de dato	Comentarios
`owner`	`string`	Dirección de correo de quien creó el evento.
`name`	`string`	Nombre del evento, entre comillas simples.
`start`	`datetime`	Fecha y hora de inicio del evento.
`finish`	`datetime`	Fecha y hora de término del evento.
`description`	`string`	Descripción del evento, entre comillas simples.
`invited`	`lista de strings`	Lista de las direcciones de correo de los invitados, separadas con ";".
`labels`	`lista de strings`	Lista de etiquetas, separadas por ";".

Cuadro 3: Estructura `db_eventos.csv`

Como podrás haber notado, el formato de los archivos son `.csv`, lo que es un acrónimo para:

"*Comma-Separated Values*"

Como podrás haber notado, pueden existir comas (",") dentro de las descripciones, nombres, títulos y cuerpo, por lo que estos textos están entre comillas simples (" ' ") las cuales no cuentan para los 256 caracteres de la encriptación. Estas comillas permiten tener texto coherentes que permitan el uso de comas, por lo que tu versión de *DCCorreos* también debe poder escribir y leer archivos en este formato sin perder coherencia del texto confundiéndolo como si fuese otra columna.

Por otra parte, también se tiene el archivo `old_db_emails.csv` el cual contiene los correos electrónicos enviados sin encriptar para que así puedas comparar y ver si estás realizando bien la encriptación.

5. Notas

- La libreria `datetime` te puede ayudar con el manejo de fechas. Su documentación se encuentra **aquí**.

6. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.6.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del foro si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. **Tendrás hasta 24 horas después del plazo de entrega** de la tarea para subir el *readme* a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la **guía de descuentos**.
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0).