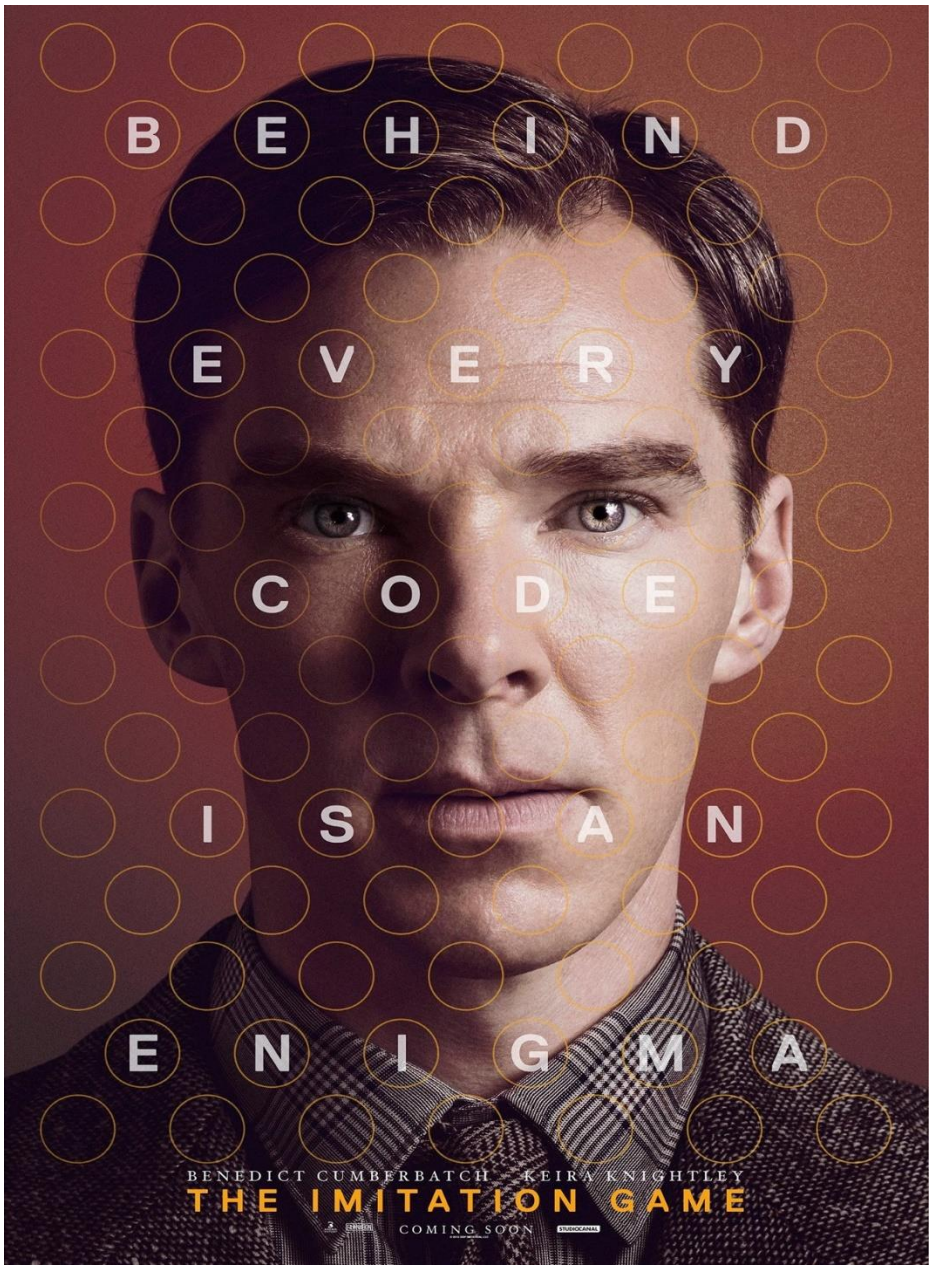
25 DE AGOSTO DE 2019



ENIGMA MACHINE

URIEL GÓMEZ REYES
PERSONAL
https://github.com/Hatxu/Enigma

Tabla de contenido

Introducción:

Si no lo saben todavía, la Maquina enigma fue una maquina de codificación de la segunda guerra mundial para codificar mensajes. La maquina Enigma fue revolucionaria y jugó un papel importante en la segunda guerra mundial y la historia detrás de ella tiene que ver con la misma computación en sí. Alan Turing es el padre de la computación moderna y si bien este documento no trata sobre él, se me hace importante mencionarlo.

Bien el documento que estas leyendo ahorita habla de la elaboración de la maquina enigma en Python 3.0, si bien no cambia relativamente mucho entre 2.7 y 3.0 se tendría que ajustar el código para que corra correctamente.

Si quieres aprender mas de la maquina Enigma y su historia este no es el mejor documento para hacerlo, este realmente es un documento de por qué y cómo replique la maquina enigma. Como dato curioso la porta es de la película “Imitation Game” habla de la vida de Alan Turing y como descifro enigma, no tengo derechos de la imagen solo quería mencionarlo.

¿Por qué?

Realmente pongo esto porque siempre que le muestro a alguien Enigma todos me hacen la misma pregunta. ¿Por qué? Cuando tuve la entrevista de Oracle y me dijeron ocupaba proyectos y si quería podía hacer uno, realmente cuando salí de la entrevista no tenía ni idea que quería hacer. Lo único que hice por un par de días fueron proyectos de programación de Euler si tienen curiosidad donde están, busquen en el repositorio de Euler.

Ahora, acabo de terminar mi Diplomado de Data Science pero en ese tiempo aun lo estaba cursando. Realmente quería hacer en su momento un predictor de bolsa de valores con Machine Learning, ya había hecho en su momento en la universidad, pero con una formula de finanzas de compra y venta de acciones. Cuando me puse a pensar en el diseño realmente me di cuenta de dos cosas, la primera es que ocupaba información para poder entrenar el modelo, y la segunda y mas importante, es que la interfaz grafica del programa seria un fastidio enorme. Si no entienden la razón es muy fácil, primero necesitaría poder descargar las acciones de alguna compañía digamos GOOGLE por ejemplo, y mostrar el movimiento de acciones en el tiempo, entrenar al modelo se supone que debiera ser una red neuronal que ya estuviera entrenada y solo tuvieras que cargar la información, pero el problema de la interfaz si alguna ves han usado programas como “Plus500” o alguno mas

local como el que usa Banxico es que tu predictor debiera mostrar una un cuanto pueden subir y bajar las acciones en 5,15,30 y 60 minutos. Hacer eso visualmente sería un fastidio, por el hecho de que talvez ocupara ingresar usuarios o talvez tener la opción de ver múltiples cuentas como Google, Facebook, y divisas como el dólar al mismo tiempo.

¿Por qué es un Fastidio? Realmente es el tiempo, quisiera decir que tengo tiempo para buscar el framework y trabajar en la interfaz gráfica, ese fue el principal problema de el primer intento de proyecto. Ahora que tiene que ver con todo esto la maquina enigma, bien hace años vi la película de “imitation game”. Cuando descarte la idea del machine learning, me puse a pensar que tan difícil sería construir yo mi propia maquina enigma. Después de hacer la investigación de como funciona enigma, me di cuenta qué con mi conocimiento actual de programación podía realmente construir algo así, y tenia dos ventajas. La primera era que es un reto de programación, como me gustan los retos realmente me sentía motivado por hacerlo, la segunda y más importante podía hacerlo en línea sin necesidad de descargar nada. Si, el secreto de este programa es que todo se hizo usando http://onlinegdb.com/ realmente le agradezco a esa pagina por tener un compilador de Python 3.0 en línea. Ahora es importante decirlo por que con el tiempo del diplomado en su tiempo el único tiempo que tenia para trabajar en el proyecto era en mis tiempos libres en el trabajo, y como no tenia Spyder o el IDLE de Python Instalado en mi maquina de trabajo funciono para mí. Ahora la razón de la construcción de enigma fue porque pude.

¿Cómo Funcona Enigma?

Cuando explicas que es enigma es sencillo si me preguntas, escribes una letra y te entrega otra. El problema de la programación de Python de enigma no es que hace enigma, si no como es que lo hace.

La imagen de arriba es una maquina enigma estándar, solo consta de las 26 letras del alfabeto inglés, siempre tuve la duda porque usaban ese tipo de alfabeto si eran alemanes y su alfabeto consta de más vocales y una letra “ß” que parece B pero suena como s.

La máquina original de Enigma funciona al presionar una letra en una máquina de escribir, encima de la máquina de escribir existe un panel con la misma cantidad de letras, esta se ilumina como respuesta. Por ejemplo, si yo presione la letra “K” en la máquina de escribir, se iluminará la letra “Q” en el panel como respuesta.

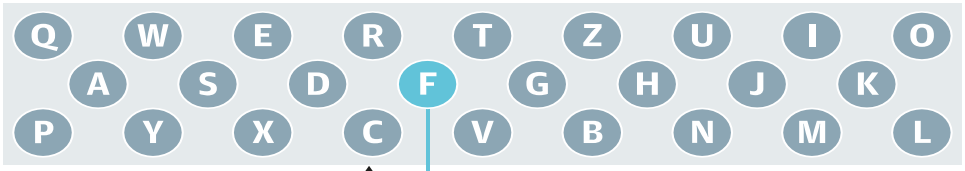
La máquina enigma consta de 3 partes importantes, Los enchufes (Plugs), Rotores y el Reflector. Explicare parte por parte en forma de código para explicar el flujo de código enigma, pero enserio si quieren ver una mejor explicación aquí hay un enlace junto a una imagen

Fuente:

- https://www.theguardian.com/technology/2014/nov/14/how-did-enigma-machine-work-imitation-game

En la parte de abajo copie la imagen para que vean mejor el funcionamiento de enemiga, es del mismo enlace. Todo está en inglés, realmente no es un problema solo lo menciono, de ahí en fuera empezare a explicar mi código clase por clase y su flujo. Dejo la imagen como punto de partida y desde este momento asumiré que entienden como funciona enigma. De todas formas, hare mi mejor esfuerzo a la hora de explicarlo dentro del código. Parte por parte.

Enigma How the machine worked

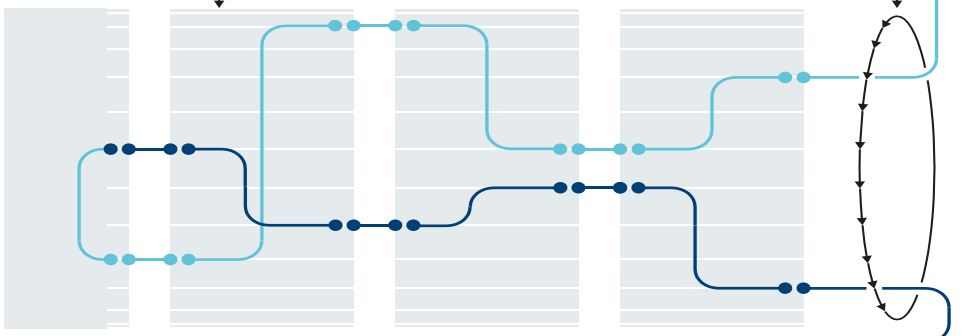


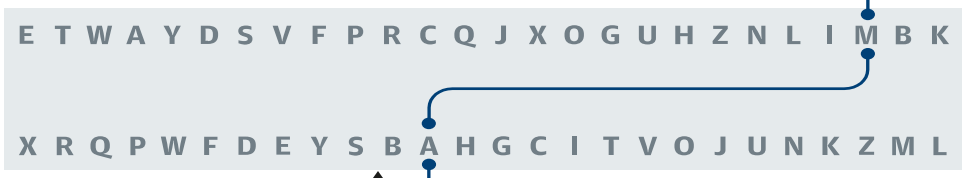
1 | Sender types each letter of the message into a **standard keyboard**

2 | Signal passes through **plugboard** which switches letters around

3 | Three physical **rotors**, internally wired like spaghetti, each change the output letter before a **reflector** sends the signal back through the whole system

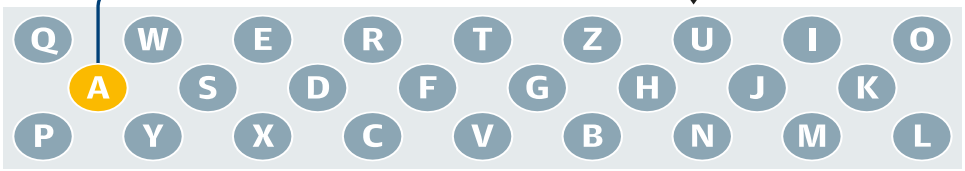
4 | To add to the code's complexity, the first rotor rotates one step **after each key press** (after 26 the second starts to move) so 'FF' might produce 'AQ'





5 | The reflected signal makes a second pass through the **plugboard**

6 | Finally, the signal **lights up a letter** on a light board



7 | Sender copies the lit letters and sends encryption using morse code. The system relies on sender and receiver **setting up** in the same pattern

8 | The receiver writes down apparent gibberish but, on keying in 'A', finds a letter F lights up and soon the encrypted message becomes clear

PAUL SCRUTON, GUARDIAN GRAPHIC

SOURCE: SIMON SINGH, LOUISE DADE

Plug o Enchufe:

Para que quede claro plug y enchufe es exactamente lo mismo, cuando investigue de enigma la mayoría de la documentación estaba en inglés, y codeé parte del código en inglés, pero al acabar el código talvez algunas funciones y la interfaz están en español. Ese fue un error de mi parte.

Esta fue la parte más fácil de hacer de enigma, realmente es hacer un objeto que guarde dos valores. Si escribo el primer valor, me entregara el segundo y si escribo el segundo valor me entregara el primero.

Ejemplo:

Al crear un enchufe le doy dos valores “A” y “X” si escribiera la palabra “HOLA” el resultado es “HOLX”. Si escribo quiero que el Resultado sea “HOLA” tengo que escribir “HOLX” por el cambio de letras. Ahora un poco de historia antes de explicar el código, los alemanes usualmente usaban hasta 12 plugs, el máximo son 13. No hay un numero establecido de cuantos puedes poner o no. Puedes no poner alguno si quieres, la razón de que pusieran es que en agrega un nivel mas de complejidad a la maquina enigma. El límite son 13 por que solo existen 26 letras, tomando en cuenta que cada enchufe tiene 2 letras y es imposible para esa maquina que un enchufe repita letras el límite son 13. Es tarde para mencionarlo pero a todo esta parte de la maquina enigma se llama “Plugboard” por eso así se llama la clase.

Bien el código es este:

```
class Plugboard:
    def __init__(self,plug1,plug2):
        self.plug1 = plug1
        self.plug2 = plug2

    def __str__(self):
        return "PlugA: "+  self.plug1 + " y PlugB: "+ self.plug2

    def changeplug(self,value):
        if value==self.plug1:
            return self.plug2
        elif value==self.plug2:
            return self.plug1
        else:
            return "0"
```

Esto es lo que decía con que el código no es congruente en el idioma, pero bueno si quieres declarar un enchufe solo debes escribir.

Plug1 = `Plugboard(“A”,”X”)`

Asi es si quisieras crear un objeto, el orden no tiene ninguna importancia da el mismo resultado si escribes eso o al revés que sería así.

Plug1 = `Plugboard(“X”,”A”)`

Por si solo el plug no funciona, la razón es que no hará nada, el siguiente código:

- Parte 1:

```
plugboardslist = []
```

- Parte 2:

```
plugboardslist.append(Plugboard(NodoA, NodoB))
```

- Parte 3:

```
def plugValor(letra):
    if len(plugboardslist)==0:
        return(letra)
    if sorted([valor.changeplug(letra) for valor in plugboardslist])[-1] !=
'0':
        return(sorted([valor.changeplug(letra) for valor in
plugboardslist])[-1])
    else:
        return(letra)
 #return sorted([valor.changeplug(letra) for valor in plugboardslist])[-1]
if sorted([valor.changeplug(letra) for valor in plugboardslist])[-1] != '0'
else letra
```

Primero que nada, si los enchufes no están creados, debes tu crearlos manualmente y segundo no sabes cuantos enchufes habrá. La primera parte es una lista vacía donde se guardarán todos los enchufes. La segunda parte, es donde se agrega un objeto a la lista, el Nodo se refiere a la variable del enchufe no a la letra. El nodo puede tener cualquier valor.

Bien la tercera parte es la función que hace todo el trabajo de decidir si la letra pertenece a un enchufe y si pertenece hacer el cambio de valor. La razón por la que encaso de no hacer

un cambio de valor regresa un 0 es para esta función, esta función se encarga de manipular a toda la lista de enchufes. En caso de no haber ningún enchufe funcionara como un teclado normal. En caso de que no, ya que todos los demás enchufes dieron 0 entregara el cambio del enchufe de la lista. Es fácil si tengo 5 enchufes y solo uno tiene la letra que quiero cambiar, esta letra será siempre la ultima letra de la lista. Si la ultima letra de la lista es 0 devuelve un valor normal.

Nota: El código comentado es toda la función escrita en una sola línea, de código, realmente no sabia que era mejor por eso deje la otra forma para que sea mas visual. Necesita una pequeña corrección ya que no abarca la parte de que la lista este vacía, ya sea que se dejara el if.

```
def plugValor(letra):
      if len(plugboardslist)==0:
            return(letra)
      return sorted([valor.changeplug(letra) for valor in plugboardslist])[-
1] if sorted([valor.changeplug(letra) for valor in plugboardslist])[-1] !=
'0' else letra
```

Así se vería sin el comentario.

Rotor:

Esta es la parte mas complicada de todo el código enigma, podría decir que el rotor es el corazón de cómo funciona enigma, este tiene un diccionario interno, y este funciono en base a la posición del rotor y la posición de la letra. Después de devolver un valor este, cambia de posición en base a su velocidad.

```
class Rotor:
    def __init__(self,modelo,posicion,velocidad):
        self.modelo = modelo
        self.posicion = posicion
        self.velocidad = velocidad
        self.lista =
["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"]
        self.diccio= {} #Diccionario Input
        self.version(modelo)
```

```
    def __str__(self):
        return "El rotor de velocidad: "+self.velocidad+", es un modelo:
"+self.modelo+", y se encuentra en la posicion: "+self.posicion

    def version(self,modelo): #Modelo del rotor
        if modelo == 'I':
            self.diccio = {'A': 'E', 'B': 'K', 'C': 'M', 'D': 'F', 'E': 'L',
'F': 'G', 'G': 'D', 'H': 'Q', 'I': 'V', 'J': 'Z', 'K': 'N', 'L': 'T',
'M':'O', 'N': 'W', 'O': 'Y', 'P': 'H', 'Q': 'X', 'R': 'U', 'S': 'S', 'T':
'P', 'U': 'A', 'V': 'I', 'W': 'B', 'X': 'R', 'Y': 'C', 'Z': 'J'}
        else:
            self.diccio=("No existe ese modelo")

    def valorabc(self,valor): #Es para sacar los valores de la tabla y
obtener un resultado.
        return self.lista.index(valor)+1

    def pasoletra(self,letra): #devuelve la letra del diccionario
        return self.diccio.get(letra)

    def letracual(self,valor): #En base a la posicion de la letra te dice
cual sigue"
        resultado = self.lista.index(self.posicion)+valor-1
        if resultado >= 26:
            return self.lista[resultado-26]
        return self.lista[resultado]
```

Nota:

Borre algunos modelos, para que cupiera mejor en el documento.

Muy bien. ¿Qué es lo que hace complejo a enigma? 3 cosas: El modelo, el numero de rotores y la posición. Los alemanes no usaban un solo rotor, la maquina enigma estándar usaba 3, existían algunas que tenían hasta 4 rotores. La razón, es que entre más rotores más encriptaba el mensaje en flujo de enigma. Cada Rotor podía ser de un modelo diferente y tener una posición distinta, esto es lo que hace complejo a enigma.

Algo que tengo que aclarar es que no declaras los rotores, ya que estos vienen ya dentro de la maquina enigma, no puedes crearlos, solo editarlos al hacerlo así me ahorro muchos problemas y restrinjo errores, de otra forma la gente debiera entender cual es la velocidad de un rotor y sintaxis de ese estilo. Solo use 3 rotores que era lo más común en las maquinas enigma, algunas llegaban a tener 4 rotores pero era muy raro,

El modelo significa un diccionario completamente distinto, me refiero que si en un diccionario entra una “A” y sale una “Q”, otro diccionario puede tener que su “A” valga una “Z”. Al tener todos los rotores juntos sin contar el movimiento podría ser un ejemplo así:

Escribo “A” – Primero Rotor ”Z” -- Segundo Rotor ”Q” – Tercer Rotor “W”

Convierte en la Z – Convierte la Z en Q – Convierte la Q en W

Si no tuviera en cuenta el movimiento por solo escribir la letra A, podría cambiar hasta la letra W. Por el hecho de los modelos del Rotor, pero esto seria como tener un diccionario gigante al final no afecta nada los cambios que haga solo es un diccionario grande.

Para explicar la siguiente parte tienes que saber que enigma no es una computadora, es una máquina, funciona con señales, esto es importante para explicar la posición de los rotores.

Piensa en el rotor como un elevador, usualmente nuestro alfabeto la letra A es la primera y la Z es la ultima letra del alfabeto, en el rotor no es así, dependiendo de su posición esto cambia.

Piensa que el Rotor esta en la posición B, entonces la posición B se vuelve la primera letra de tu alfabeto y la letra A se vuelve en la ultima letra de tu alfabeto.

```
    def letracual(self,valor): #En base a la posicion de la letra te dice
cual sigue"
        resultado = self.lista.index(self.posicion)+valor-1
        if resultado >= 26:
            return self.lista[resultado-26]
        return self.lista[resultado]
```

Eso es exactamente lo que hace este método dentro de la clase, cuando tu ingresas una letra te dice que letra en el nuevo alfabeto eres y luego se usa este otro método para darte su valor del diccionario.

```
def pasoletra(self,letra): #devuelve la letra del diccionario
        return self.diccio.get(letra)
```

Lo que estoy diciendo no lo hace la clase por si sola. Para eso es esta función externa. Esto es solo por 1 solo rotor, hace el cambio en base a la letra y diccionario y posición del rotor.

```
print("R1i: ",rotor1.letracual(rotor1.valorabc(rotor1.pasoletra(
plugValor(valor) ))) )
```

Nota:

Este ya toma parte del flujo de los rotores que es los plugs o enchufes y no se ve tal cual, dentro del código, es un comentario. Mas de esto adelante.

```
def movimientoRotores(valor):

lista=["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"]
    if rotor1.posicion == "Z":
        rotor1.posicion = lista[0]
        if rotor2.posicion == "Z":
            rotor2.posicion = lista[0]
            if rotor3.posicion == "Z":
                 rotor3.posicion = lista[0]
            else:
                rotor3.posicion =lista[lista.index(rotor3.posicion)+1]
        else:
            rotor2.posicion = lista[lista.index(rotor2.posicion)+1]
    else:
        rotor1.posicion = lista[lista.index(rotor1.posicion)+1]
    return señalRes(valor)
```

Esta función se encarga de mover la posición de los rotores basado en el rotor y su velocidad, esto hace que después de cada tecla cambie de valor la posición del rotor. Es algo bastante sencillo si el rotor no es Z avanza una letra en el diccionario, y si llega a “Z” lo reinicia a la letra “A”. Al cambiar a la letra “A” en un rotor mueve una posición en el siguiente rotor, es por eso que tienes 3 ifs. Es algo bastante sencillo y funciona en actualizar los rotores, Esta es la razón de que abuse de que era 3 rotores fijos, si se pudiera manipular el número de rotores, tendría que cambiar la estructura de esta función.

Los modelos como ya dije son solo diccionarios, en la parte de evidencia vienen otras versiones de diccionarios, me base en los 5 modelos mas utilizados, pero se pueden agregar mas.

Reflector

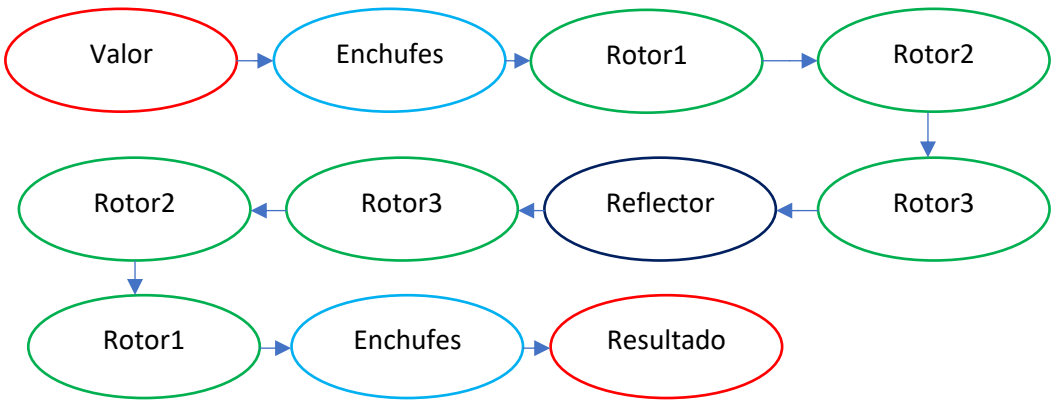
Deje en teoría lo más fácil al final, el reflector es solo un diccionario, y esto realmente es un rotor, sin modelo o posición, solo recibe una señal y la regresa. Ahora viendo el código, podría haber heredado el rotor del reflector y usar todo, pero la razón es que hice primero el rotor antes del reflector, y solo borre los métodos y variables. Como su función también es distinta lo deje como una clase aparte en ves de el la clase padre. Viendo el código entenderán a lo que me refiero.

```
class Reflector:
    def __init__(self,modelo):
        self.modelo = modelo
        self.diccio= {}
        self.version(modelo)

    def __str__(self):
        return "El Reflector es un modelo: "+self.modelo

    def version(self,modelo): #Modelo del Reflector
        if modelo == 'UKW-A':
            self.diccio = {'A': 'E', 'B': 'J', 'C': 'M', 'D': 'Z', 'E': 'A',
'F': 'L', 'G': 'Y', 'H': 'X', 'I': 'V', 'J': 'B', 'K': 'W', 'L': 'F',
'M':'C', 'N': 'R', 'O': 'Q', 'P': 'U', 'Q': 'O', 'R': 'N', 'S': 'T', 'T':
'S', 'U': 'P', 'V': 'I', 'W': 'K', 'X': 'H', 'Y': 'G', 'Z': 'D'}
        else:
            self.diccio=("No existe ese modelo")

    def pasoletra(self,letra): #devuelve la letra del diccionario
        return self.diccio.get(letra)
```

Es como lo dije, es lo mismo del rotor, pero sin velocidad, y posición.

Flujo:

Pensando que entendieron ya como funciona el flujo de enigma, se los repetiré;



Muy bien teniendo esto en cuenta es la razón de esta línea de código.

```
def señalRes(valor):
          return
plugValor(rotor1.letracual(rotor1.valorabc(rotor1.pasoletra(
rotor2.letracual(rotor2.valorabc(rotor2.pasoletra(
rotor3.letracual(rotor3.valorabc(rotor3.pasoletra( reflector.pasoletra(
rotor3.letracual(rotor3.valorabc(rotor3.pasoletra(
rotor2.letracual(rotor2.valorabc(rotor2.pasoletra(
rotor1.letracual(rotor1.valorabc(rotor1.pasoletra(plugValor(valor)))) )))
))) ) ))) ))) ))) )
```

Esta línea de código hace exactamente ese flujo, tal vez pude haber juntado los métodos del rotor para hacerlo más corto, pero así realmente funciona el código.

Nota:

Esta función tiene código comentado, que demuestra cada una de las etapas del cifrado.

Interfaz

Muy bien esta es mi justificación de porque esa interfaz. Cuando cree a enigma lo hice en un compilador web, realmente se me hacia justo que pudiera funcionar en un compilador web también. Es por eso que tiene una interfaz de comando antigua, realmente este es el enigma 1.0 Si tengo pensando hacerle una interfaz gráfica, pero me surgió otro proyecto, por eso por el momento se quedo como enigma 1.0.

La interfaz tiene un flujo bueno si sabes como funciona. No copiare el código de la interfaz porque es la mas extenso de todo el código. Cuenta con validaciones, y protección para evitar equivocarse. Algo que tengo que mencionar es cuando le agarras el rollo es rápido, pero lo único con lo que no cuenta es con la edición de enchufes, esto es porque en este tipo de interfaz poder editar los enchufes ya creados en la lista iba a ser demasiado tardado, por eso opte por no editarlos, y dejarlo para la versión grafica 2.0 fuera de esto y una función de explicación para cada sección esta versión funcionaria como la esperada den la versión 2.0.

El mapa de la interfaz es el siguiente:

Enigma

1. Configuración Enigma
    - 1.1 Opciones Enchufes
        - 1.1.1 Crear Enchufes
        - 1.1.2 Ver Enchufes
        - 1.1.3 Volver Atrás
        - 1.1.4 Volver al menú principal
    - 1.2 Opciones Rotores
        - 1.2.1 Editar Rotores
            - 1.2.1.1 Editar Rotor Rápido
            - 1.2.1.2 Editar Rotor Medio
            - 1.2.1.3 Editar Rotor Lento
            - 1.2.1.4 Volver Atrás
            - 1.2.1.5 Volver al menú Principal
        - 1.2.2 Ver Rotores
        - 1.2.3 Volver Atrás
        - 1.2.4 Volver al menú principal
    - 1.3 Opciones Reflectores
        - 1.3.1 Editar Reflector
        - 1.3.2 Ver Reflector
        - 1.3.3 Volver Atrás
        - 1.3.4 Volver al menú principal
    - 1.4 Volver al menú principal
2. Usar Enigma
    - 2.1 Enigma Rápido
    - 2.2 Enigma Lento
3. Información Enigma
4. Salir

Ese es el flujo de los menús, cuando lo usas un par de veces ya es bastante rápido.

Enigma Lento:

¿Porque hablo de Enigma lento antes del rápido? La razón es muy simple, Cuando salió enigma fue creado, cifraba letra por letra. Esta Función hace exactamente eso, mi error fue que cuando la hice la incluí en la interfaz y no en una función externa. Si tengo tiempo tal ves modifique en el futuro esto.

```
mensaje = []
                    cifrado = []
                    letra = ""
                    print("Para salir escribe 'Salir'.")
                    while letra != "Salir":
```

```
                    print("Rotores: ",
rotor3.posicion,rotor2.posicion,rotor1.posicion)
                    while True:
                      try:
                        letra = input("Escribe letra: ")
                        if letra == '':
                          raise Exception
                        break
                      except Exception:
                        print("No aceptamos nulls aquí")
                    if letra == "Salir":
                        print("Mensaje Original: ","".join(mensaje))
                        print("Mensaje Cifrado:  ","".join(cifrado))
                        pass
                    else:
                      letra = letra[-1]
                    if letra.upper() in Alphabet:
                        mensaje.append(letra)
                        x = movimientoRotores(letra.upper())
                        cifrado.append(x)
                        print("Letra cifrada: ",x)

                    else:
                        mensaje.append(letra)
                        cifrado.append(letra)
```

Esto función fue creada para cifrar una letra a la vez, el motivo de una función tan ortodoxa es que hace honor a enigma realmente, ves los cambios del rotor y ves como funcionaba en su época una maquina enigma, no tiene sentido realmente en la era moderna, pero es un buen recuerdo a tener.

Restricciones:

Ya que no use un keyPress action como tenia previsto, lo deje también para la versión 2.0 para solucionar mi predicamento use inputs. Si yo escribo la palabra “HOLA”, solo tomara en cuenta la prima letra del input en este caso la “H” y la cifrara. La única excepción a esta regla es la palabra “Salir”. Acepta otros caracteres, pero estos no afectaran en el cifrado de enigma, esto es para poder escribir preguntas o separar por texto oraciones. Se me hizo agradable tener esta función y si quitas el código comentado de la función señalRes(valor): Te das cuenta de todo el trabajo que hace enigma.

Nota:

No he probado el código sin documentar en la función rápida y realmente no lo recomiendo intentar.

Enigma Rapido

Esta función cifra texto completo, si escribes una palabra o oración digamos “Hola como estas.” Cifrara por completo la oración sin restricciones.

```
def mensajerapido(mensaje): #Modo Veloz
    codigo = list(mensaje)
    return "".join([movimientoRotores(i.upper()) if i.upper() in Alphabet
else i for i in codigo])
```

Nota:

Ambas versiones funcionan con mayúsculas y minúsculas, por lo que no habría problema, el resultado siempre será en mayúsculas.

Conclusión:

Este realmente es un código de diversión tiene muchísima historia de trasfondo y cuando lo usas te das cuenta de que el tipo que diseño a enigma para su época era increíble. Tanto el como Turing por desencriptarlo eran asombrosos.

Versiones:

Este código viene con versiones, no todas funcionan, realmente muestra la evolución de enigma.

1. https://www.onlinegdb.com/ryx_tvigs
2. https://www.onlinegdb.com/sjyedgymr
3. https://www.onlinegdb.com/sygfab0zb
4. https://www.onlinegdb.com/rylm3lx4s
5. https://www.onlinegdb.com/ryvi3u7vr
6. https://www.onlinegdb.com/s1ukhq4es
7. https://www.onlinegdb.com/rycce3_nh
8. https://www.onlinegdb.com/hk-fbndvs
9. https://www.onlinegdb.com/rjs8atteh

La versión 9, corre por si sola, no está divida en objetos como en esta y creo que tienes unos bugs menores que no fueron, ni serán corregidos. Un bug que me acuerdo de esa versión es que si escribes un null en el enigma lento tronara el programa. Recomiendo ampliamente descargar la versión del repositorio.

Bibliografía:

Me base en estos 3 documentos para crear mi maquina enigma.

https://web.stanford.edu/class/cs106j/handouts/36-TheEnigmaMachine.pdf

https://www.cryptomuseum.com/crypto/enigma/wiring.htm

https://en.wikipedia.org/wiki/Enigma_machine