IMPLEMENTACIÓN DE LA MÁQUINA ENIGMA

Subtítulo de la presentación

CLASES UTILIZADAS:

- Máquina
- Rotor
- Plugboard

CLASE ROTOR

ATRIBUTOS

```
class Rotor:
    #permutacion de rel en cada giro
    dest_go: str
    count: int
```

- dest_go: string que contiene los 26 caracteres del alfabeto.
- count: entero que cuenta el número de veces que el rotor ha girado a causa de presionar los botones de la máquina

INSTANCIACIÓN DE LA CLASE ROTOR

```
def __init__(self, char_init='A', dest_go=""):

    self.count = 0
    self.dest_go = dest_go

    if(char_init != 'A'):
        ind_init = ascii_uppercase.find(char_init)
        self.dest_go = self.dest_go[ind_init:] + self.dest_go[:ind_init]

    if(len(dest_go) == 0):
        self.dest_go = self.set_random_string()
```

- El constructor recibe dos argumentos, el caracter inicial y una cadena de 26 caracteres que representará el cableado interno del rotor.
- Vemos que si la posición inicial ha sido modificada, el cableado interno del rotor también se verá afectado.

EJEMPLO DE INSTANCIACIÓN DE UN ROTOR

```
aux_go = "BDFHJLCPRTXVZNYEIWGAKMUSQO"
letra_inicial = 'B'
rt = Rotor(letra_inicial , aux_go)
```

MÉTODOS DE LA CLASE ROTOR

- move(): modifica el cableado cuando el rotor se mueve una posición. Se llamará en la clase **MÁQUINA** cuando se presiona una tecla.

```
def move(self):
    aux_dest_go = self.dest_go[0]
    self.dest_go = self.dest_go[1:] + aux_dest_go
```

- push(char): recibe un caracter y te retorna el caracter al que está cableado. Se usa en el flujo de ida del a encriptación.

```
def push(self, char_to_encrypt):
    new_char = self.dest_go[ascii_uppercase.find(char_to_encrypt)]
    return new_char
```

- antipush(char): procesa una letra que ingresa en el flujo de vuelta de la encriptación tras pasar por el reflector.

```
def antipush(self, reflected_char):
    new_char = ascii_uppercase[self.dest_go.find(reflected_char)]
    return new_char
```

LA CLASE PLUGBOARD

ATRIBUTOS:

- Esta clase solo tiene como atributo un diccionario que le permite modelar el cableado en el panel frontal de una máquina enigma real.

```
class Plugboard:

    cableado : dict


    def __init__(self, cableado = {}):
        self.cableado = cableado
```

ATRIBUTOS

- Esta clase solo tiene como atributo un diccionario que le permite modelar el cableado en el panel frontal de una máquina enigma real.

```
class Plugboard:

    cableado : dict


    def __init__(self, cableado = {}):
        self.cableado = cableado
```

INSTANCIACIÓN DE LA CLASE PLUGBOARD

- Para instanciar esta clase solo es necesario pasarle al constructor un diccionario que simule el cableado del panel frontal.

```
cables = {
'A' : 'C',
'B' : 'F',
'G' : 'H',
'L' : 'N',
'K' : 'R',
'Q' : 'E'
}


plugboard = Plugboard(cables)
```

MÉTODOS

- **procesar_caracter (char):** este método solo recibe un argumento de tipo CHAR que viene a ser el carácter que el Plugboard va a procesar

```
def procesar_caracter(self, char):
    char_procesado = ''

    cableado_claves = list(self.cableado.keys())
    cableado_valores = list(self.cableado.values())

    if char in cableado_claves:
        char_procesado = self.cableado[char]
    elif char in cableado_valores:
        char_procesado = cableado_claves[cableado_valores.index(char)]
    else:
        char_procesado = char

    return char_procesado
```

LA CLASE MÁQUINA

ATRIBUTOS:

- **list_of_rotors**: este atributo es un arreglo que almacenará objetos de la clase Rotor representando a todos los rotores de la máquina. La máquina solo podrá tener hasta 5 rotores.
- **Plugboard:** este atributo es un objeto de la clase Plugboard.
- **Reflector:** este atributo representará al reflector mediante un objeto de la clase Rotor.

```
class Maquina:
    list_of_rotors: list
    reflector: Rotor
    plugboard: Plugboard

    def __init__(self, nmrRotors, initCharRotors, cables):

        self.list_of_rotors = []
        reflector_indexes = "IXUHFEZDAOMTKQJWNSRLCYPBVG"

        #rotors_orders = ["EKMFLGDQVZNTOWYHXUSPAIBRCJ", "AJDKSIRUXBLHWTMCQGZNPYFVOE", "BDFHJLCPRTXVZNYEIWGAKMUSQO"]
        rotors_orders = ["EKMFLGDQVZNTOWYHXUSPAIBRCJ", "AJDKSIRUXBLHWTMCQGZNPYFVOE", "BDFHJLCPRTXVZNYEIWGAKMUSQO", "RMSLNIXCYPTZAHJOUFDBQGWKVE", "ZIHFQEM

        # "RMSLNIXCYPTZAHJOUFDBQGWKVE", "ZIHFQEMASYPOWDBKVXCNLRTUGJ"
        # imaginemos nmrRotor = [2,1] y charRotor = ['C','G']

        for index, (nmrRotor, charRotor) in enumerate(zip(nmrRotors, initCharRotors)):
            self.list_of_rotors.append(Rotor(charRotor, rotors_orders[nmrRotor-1]))

        self.plugboard = Plugboard(cables)
        self.reflector = Rotor('A', reflector_indexes)
```

INSTANCIACIÓN

- En el constructor se tienen cadenas predefinidas con los cableados de los 5 rotores y del reflector.

```
reflector_indexes = "IXUHFEZDAOMTKQJWNSRLCYPBVG"

#rotors_orders = ["EKMFLGDQVZNTOWYHXUSPAIBRCJ", "A
rotors_orders = ["EKMFLGDQVZNTOWYHXUSPAIBRCJ",
"AJDKSIRUXBLHWTMCQGZNPYFVOE",
"BDFHJLCPRTXVZNYEIWGAKMUSQO",
"RMSLNIXCYPTZAHJOUFDBQGWKVE",
"ZIHFQEMASYPOWDBKVXCNLRTUGJ"]
```

- El constructor tiene como argumentos una lista con los rotores que se desean utilizar, una lista con las posiciones iniciales de cada uno de los rotores y un diccionario para el plugboard.

```
rotors_numbers = [2,1,3,5,4]
init_chars = ['C','F','G','B','M']
cables = {
'A' : 'C',
'B' : 'F',
'G' : 'H'
}

mq = Maquina(rotors_numbers, init_chars, cables)
```

MÉTODOS

- moveListOfRotors(): este métodose usará despues de cifrar un caracter. Empezará moviendo el primer rotor y cada uno de los siguientes rotores se moverá solo si su anterior ha realizado 26 movimientos

```
def moveListOfRotors(self):
    for index,rotor in enumerate(self.list_of_rotors):
        if index == 0:
            rotor.move()
            rotor.count += 1
        else:
            if self.list_of_rotors[index-1].count == 26:
                self.list_of_rotors[index-1].count = 0
                rotor.move()
                rotor.count += 1
```

- cifrar(char): este método recibe como argumento un caracter que procederá a cifrar. El método hará que el caracter sea procesado por el plugboard, los rotores elegidos, el reflector, nuevamente por los rotores en sentido contrario y finalmente por el plugboard. Finalmente y de ser posible, cada rotor se moverá una posición.

```
def cifrar(self, char):
    char_processed = char

    char_processed = self.plugboard.procesar_caracter(char_processed)

    for rotor in self.list_of_rotors:
        char_processed = rotor.push(char_processed)

    char_processed = self.reflector.push(char_processed)

    for rotor in self.list_of_rotors[::-1]:
        char_processed = rotor.antipush(char_processed)

    char_processed = self.plugboard.procesar_caracter(char_processed)

    self.moveListOfRotors()
```