

# Reporte semanal - Semana 6

08/03/2024

—

Ismael Montoro Peñasco  
Fundación Goodlob

<b>Acciones llevadas a cabo.....</b>	<b>2</b>
<b>Avances realizados.....</b>	<b>2</b>
Labor 1: Encriptaciones.....	2
Labor 1.2: Corrección de estructura.....	2
Labor 2: Codificaciones.....	3
Labor 1.2: Corrección de codificaciones.....	3
Labor 3: Decodificaciones.....	4
Labor 4: Excel.....	5
Labor 5: Scapy.....	10
<b>Planes para la próxima semana.....</b>	<b>10</b>

## Acciones llevadas a cabo

Terminar con los ejercicios de hashes y encriptación, y seguir con Scapy y con Excel.

## Avances realizados

### Labor 1: Encriptaciones

He terminado de implementar en el proyecto los siguientes algoritmos de encriptado.

- TripleDES
- Blowfish
- Serpent
- RC4
- Cifrado Cesar
- Cifrado Polibio
- Cifrado Vigenere
- Cifrado Alberti

### Labor 1.2: Corrección de estructura

He corregido la estructura de mi proyecto, por la parte del cifrado, dado que había mucha redundancia de código que se podía resumir con ciertos recursos y proyectos de software de origen Alemán. Concretamente se ha podido eliminar código Java redundante, funciones de Python que hacían casi la misma función.

```

294 class cifrado_aplicado():
551
552     def descifrar_polibio(texto_cifrado: str):
553         # https://es.wikipedia.org/wiki/Cuadrado_de_Polibio
554         abecedario = {'A': 11, 'B': 12, 'C': 13, 'D': 14, 'E': 15,
555                       'F': 21, 'G': 22, 'H': 23, 'I': 24, 'J': 24,
556                       'K': 25, 'L': 31, 'M': 32, 'N': 33, 'O': 34,
557                       'P': 35, 'Q': 41, 'R': 42, 'S': 43, 'T': 44,
558                       'U': 45, 'V': 51, 'W': 52, 'X': 53, 'Y': 54,
559                       'Z': 55}
560         abecedario_clave = list(abecedario.keys())
561         abecedario_valores = list(abecedario.values())
562         texto_descifrado = ""
563
564         for i in range(0, len(texto_cifrado), 2):
565             posicion = texto_cifrado[i] + texto_cifrado[i+1]
566             posicion = int(posicion)
567             posicion_letra = abecedario_valores.index(posicion)
568             texto_descifrado += abecedario_clave[posicion_letra]
569
570         return texto_descifrado
571
572     def cifrado_alberti(self, texto: str, clave: int, rotacion: str = "derecha"):
573         """
574         Este cifrado fue creado por Leon Battista Alberti creador del criptoanálisis
575         sustitucion poli alfabetica. Tradicionalmente este cifrado se hace con un
576         mecanismo de dos discos un disco fijo y otro movil, se ajusta el movil para
577         que coincida sus caracteres con los caracteres del disco fijo,
578         posicionando el disco movil como queramos.
579
580         https://www.youtube.com/watch?v=k1Dq0c1CzQw
581         https://www.youtube.com/watch?app=desktop&v=J7m-bTK5TiE
582         """
583         rueda_fija = "DX1CGKB8M4RFUIY2L6ÑSHQ5W7EV9N3PZT3AO"
584         rueda_movil = "yqva!x0jt@&gzw&eoph%ñdr=cs?m#nfkulib"
585         numero_letras = len(rueda_movil)
586         resultado = ""
587
588         comparacion = rotacion.lower()
589         if comparacion == "derecha":
590             for _ in range(clave):
591                 ultima_letra = rueda_movil[numero_letras - 1]
592                 sub_cadena = rueda_movil[:numero_letras - 1]
593                 rueda_movil = ultima_letra + sub_cadena
594             elif comparacion == "izquierda":
595                 for _ in range(clave):
596                     primera_letra = rueda_movil[0]
597                     sub_cadena = rueda_movil[1:]
598                     rueda_movil = sub_cadena + primera_letra

```

## Labor 2: Codificaciones

Se ha podido aplicar codificaciones a objetos convertidos en bytes, concretamente a imágenes JPG/JPEG y clases de Python.

- Codificar imágenes
- Codificar objetos

```

1  import base64
2  import py3base92 as base92
3  from PIL import Image
4  import pickle, io
5
6  class codificacion_objetos():
7
8      def codificar_imagenes(self, ruta: str) -> str:
9          # Abre la imagen con PIL
10         image_pil = Image.open(ruta)
11
12         # Guarda la imagen en un objeto BytesIO
13         buffer = io.BytesIO()
14         image_pil.save(buffer, format="JPEG")
15
16         # Codifica los bytes en base64
17         image_base64 = base64.b64encode(buffer.getvalue()).decode("utf-8")
18         return image_base64
19
20     def codificar_clases(self, clase: object):
21         # Serializar la instancia en bytes
22         objeto_bytes = pickle.dumps(clase)
23         return base64.b64encode(objeto_bytes).decode("utf-8")
24

```

### Labor 1.2: Corrección de codificaciones

He corregido algunas funciones de codificación, que tenían varios fallos de lógica, como por ejemplo a la codificación BASE62.

```

def codificar_base62(self, texto = "hello"):
    BASE62 = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    # Convierte el texto en una cadena de bytes
    text_bytes = texto.encode("utf-8")

    # Convierte los bytes en un número entero
    num = int.from_bytes(text_bytes, byteorder="big")

    # Codifica el número en base62
    s = ""
    while num > 0:
        num, r = divmod(num, 62)
        s = BASE62[r] + s
    return s

```

## Labor 3: Decodificaciones

He podido realizar encontrar o implementar los algoritmos de decodificación para cada algoritmo de codificación.

- Decodificación decimales
- Decodificación hexadecimales
- Decodificación binarios.
- Decodificación base32, base45, base58, base62, base64, base85, base92.

```
class decodificacion_datos():

    def decodificar_binario(self, binario: list) -> str:
        conversion = [chr(int(byte, 2)) for byte in binario]
        return "".join(conversion)

    def decodificar_decimal(self, decimales: list) -> str:
        return "".join([chr(caracter) for caracter in decimales])

    def decodificar_base16(self, hexadecimal: str) -> str:
        codificacion = hexadecimal.encode()
        paso_bytes = bytes(codificacion)
        return base64.b16decode(paso_bytes).decode("UTF-8")

    def decodificar_base32(self, codificacion: str) -> str:
        codificacion = codificacion.encode()
        paso_bytes = bytes(codificacion)
        return base64.b32decode(paso_bytes).decode("UTF-8")
```

También he implementado los algoritmos para decodificar imágenes y objetos.

```
class decodificacion_objetos():

    def decodificar_imagenes(self, codificacion: str, ruta_nueva_guardar_imagen: str):
        # Decodifica el base64
        decoded_bytes = base64.b64decode(codificacion)

        # Crea una nueva imagen desde los bytes decodificados
        restored_image = Image.open(io.BytesIO(decoded_bytes))

        # Guarda la imagen restaurada
        restored_image.save(ruta_nueva_guardar_imagen)

    def decodificar_clases(self, codificacion: str) -> object:
        # Decodificar el base64
        decoded_bytes = base64.b64decode(codificacion)

        # Restaurar la instancia desde los bytes decodificados
        objeto_restaurado = pickle.loads(decoded_bytes)

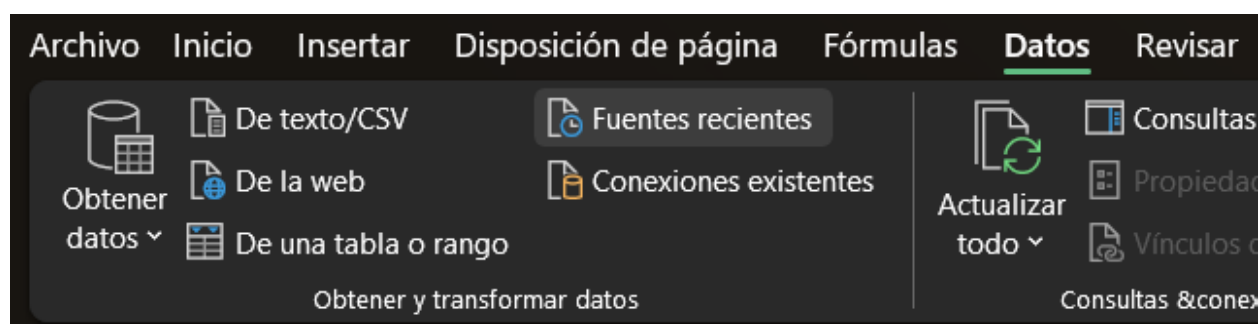
        return objeto_restaurado
```

## Labor 4: Excel

He investigado un poco las funcionalidades de Excel:

### Fuentes recientes

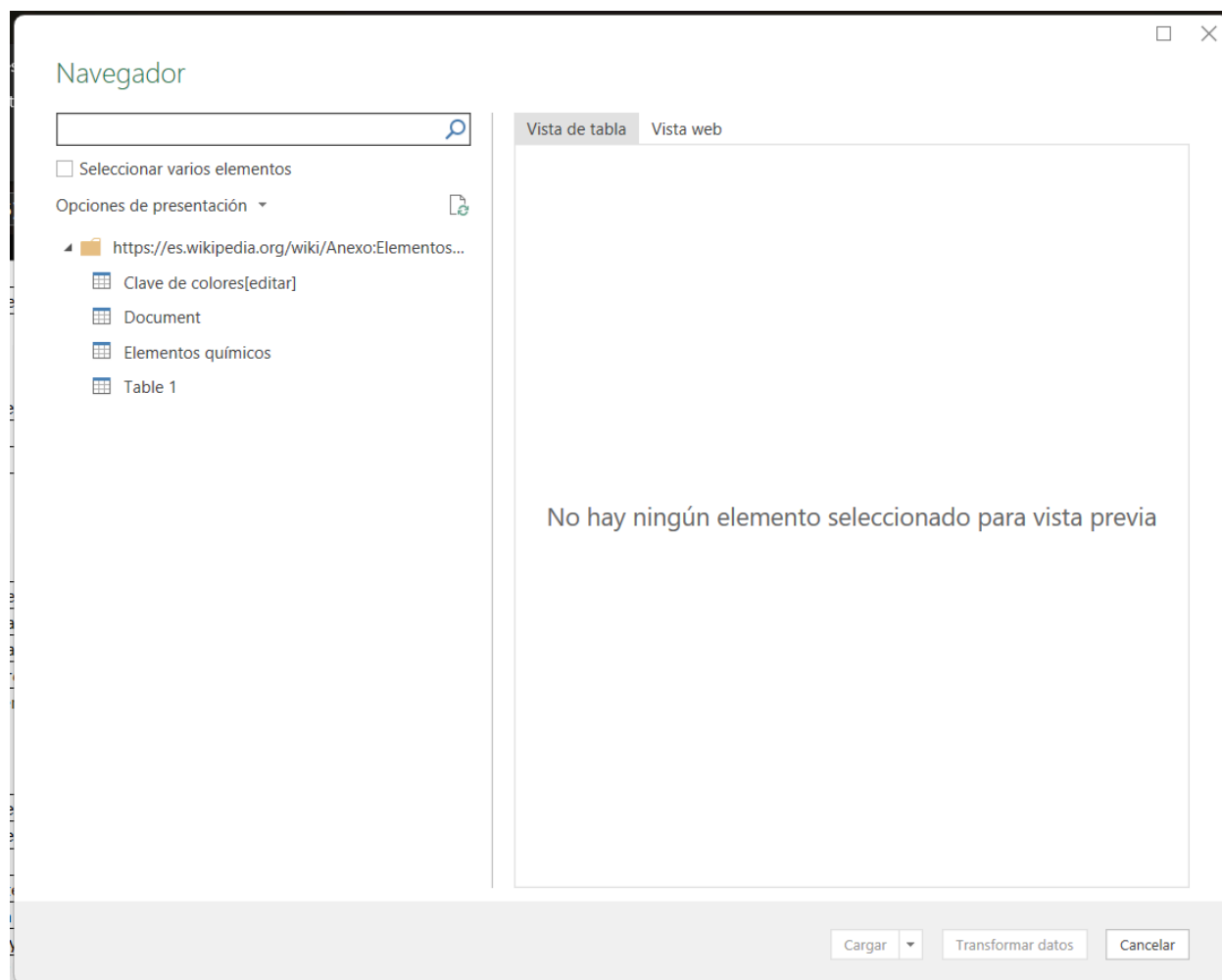
Fuentes recientes me permiten ver el historial de webs donde he extraído datos.



Si pinchamos en cualquiera de las fuentes.

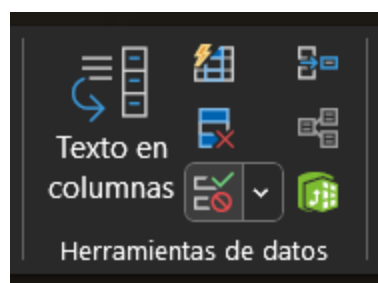


Se nos aparece esta lista, donde podemos seleccionar los datos a insertar.



## Validación de datos

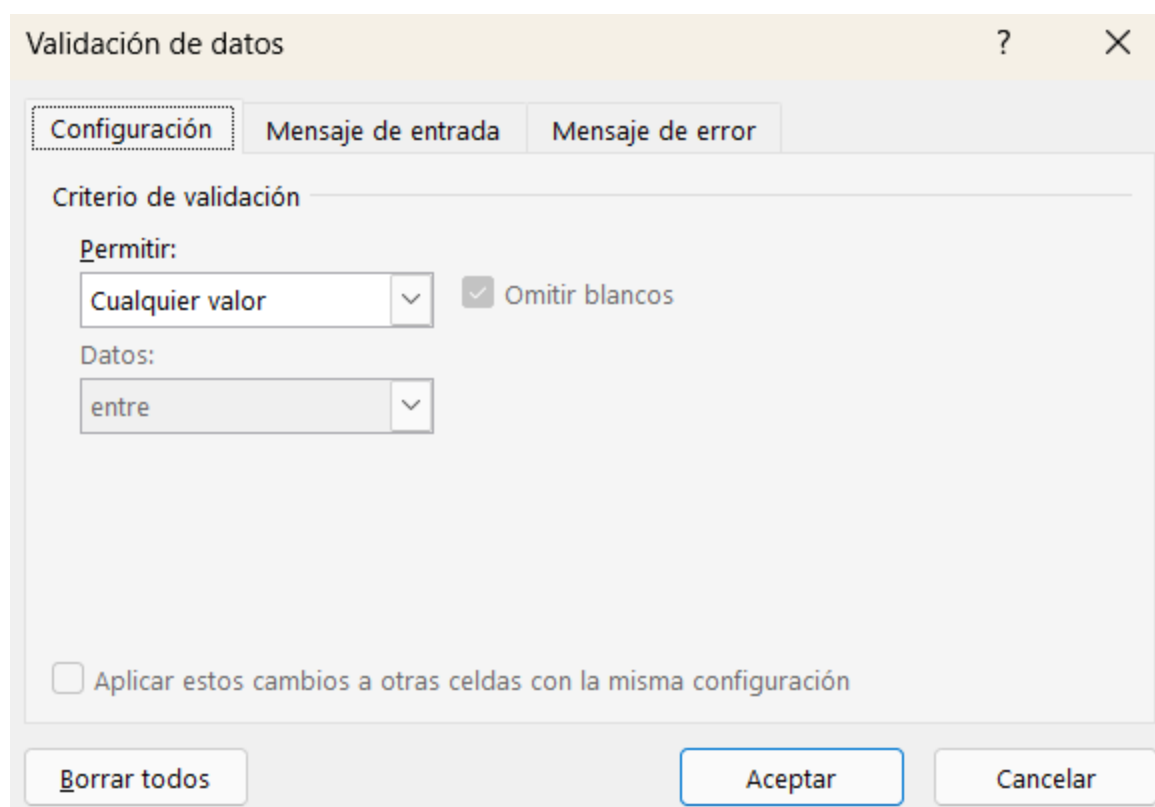
También he mirado la herramienta de validación de datos, esta me permite dentro de una celda, especificar qué valores quiero para esa celda



	A	B	C	D	E	
1						
2		Z	Símbolo	Elemento	Grupo	Periodo
3		1	H	Hidrógeno	1	
4		2	He	Helio	18	
5		3	Li	Litio	1	
6		4	Be	Berilio	2	
7		5	B	Boro	13	
8		6	C	Carbono	14	
9		7	N	Nitrógeno	15	
10		8	O	Oxígeno	16	
11		9	F	Flúor	17	
12		10	Ne	Neón	18	
13		11	Na	Sodio	1	
14		12	Mg	Magnesio	2	
15		13	Al	Aluminio	13	
16		14	Si	Silicio	14	
17		15	P	Fósforo	15	

Si tenemos una tabla con datos, podemos seleccionar una celda y pulsar en la opción de validación de datos





Validación de datos

Configuración    Mensaje de entrada    Mensaje de error

Criterio de validación

Permitir:

Cualquier valor    ☒ Omitir blancos

Datos:

entre

☐ Aplicar estos cambios a otras celdas con la misma configuración

Borrar todos    Aceptar    Cancelar

Aparece una ventana.

Validación de datos

Configuración    Mensaje de entrada    Mensaje de error

Criterio de validación

Permitir:

Lista    ☒ Omitir blancos

Datos:

entre    ☒ Celda con lista desplegable

Origen:

=D\$3:D\$19

☐ Aplicar estos cambios a otras celdas con la misma configuración

Borrar todos    Aceptar    Cancelar

En esta ventana si especificamos que queremos una lista, y qué valores queremos que estén permitidos dentro de nuestra lista, a través de la selección de valores de una columna de la tabla.

	A	B	C	D
1				
2		Z	Símbolo	Elemento
3		1 H		Hidrógeno
4		2 He		Helio
5	Hidrógeno	3 Li		Litio
6	Helio	4 Be		Berilio
7	Litio	5 B		Boro
8	Berilio	6 C		Carbono
9	Boro	7 N		Nitrógeno
10	Carbono	8 O		Oxígeno
11	Nitrógeno	9 F		Flúor
12	Oxígeno	10 Ne		Neón

Podemos crear una lista de valores para una celda.

## Labor 5: Scapy

He podido enviar y recibir paquete IP, usando una dirección DNS.

```
8
9  p = IP(dst="github.com")/ICMP()
10 r = sr1(p)
11 respuesta = sniff(count=10, prn=lambda x: x.summary())
12 for i in respuesta:
13     print(i.show())
14
```

## Planes para la próxima semana

Seguiré un poco más con los encriptados, para ir en serio con Scapy (Para realizar OSINT a servidores públicos), y voy a ir poco a poco con Excel (Para ver qué es lo que quieren las empresas concretamente de aquellos que manejes Excel). Pero lo importante es finalizar con los encriptados para poder empezar con el [curso de Inteligencia Artificial de la universidad de Harvard](#).