DOCUMENTACIÓN AVANCE 2

- Métodos de extracción de datos y herramientas empleadas.

  En la extracción de datos usamos expresiones regulares para verificar que la categoría existe al momento de pedírsela al usuario, para verificar que no hubo error según lo que dice la API y para validar que el ID sea un número.

  -Para verificar que la categoría que da el usuario existe:

```
31
32 #expresiones regulares para verificar que la categoria existe
33 DetectorCategoria = re.compile(r'any|misc|spooky|programming')
34 i = str(input("Ingresa la categoría (any, misc, spooky, programming):  "))
35 CategoriaValida = DetectorCategoria.findall(i)
36 while len(CategoriaValida) == 0:
37     print("Esa categoría no existe o la escribiste mal")
38     i = str(input("Ingresa la categoría (misc, spooky, programming):  "))
39     CategoriaValida = DetectorCategoria.findall(i)
40
```

  Creamos la variable DetectorCategoria y guardamos lo que resulta de re.compile que es lo que compila la expresión regular que ponemos, y la expresión que pusimos fue 'any|misc|spooky|programming', pusimos eso pensando en que se va a buscar cualquiera de esas 4 palabras, por la "|" que funciona como una "o", luego en CatgoriaValida usamos la variable pasada junto con .findall() y anexamos lo que hay en la variable "i" que es básicamente lo que ingresa el usuario como str, el resultado de esto es una lista con lo que encontró la expresión regular. Por último hacemos un while que se ejecuta siempre y cuando la longitud de la lista resultante sea igual a 0, ya que esto significaría que la lista esta vacía (o sea que la expresión regular no se encontró) en otras palabras, el usuario no ingreso algo válido.

  -Para verificar que no hubo error según la API

  Para formular la expresión regular que usamos, vimos la manera en la que la API devuelve la categoría de error y es exactamente 'error: False' lo que significa que no hubo error al ejecutarse

```
[{'error': False, 'category': 'Programming', 'type': 'twopart', 'setup': 'Hey baby I wish your name was asy
nchronous...', 'delivery': "... so you'd give me a callback.", 'flags': {'nsfw': False, 'religious': False,
'political': False, 'racist': False, 'sexist': False, 'explicit': False}, 'id': 53, 'safe': True, 'lang': '
en'}, {'error': False, 'category': 'Programming', 'type': 'twopart', 'setup': 'why do python programmers we
ar glasses?', 'delivery': "Because they can't C.", 'flags': {'nsfw': False, 'religious': False, 'political'
: False, 'racist': False, 'sexist': False, 'explicit': False}, 'safe': True, 'id': 293, 'lang': 'en'}]
```

```
#expresiones regulares para verificar que no hubo error
chistes2 = chistes.copy()
texto = str(chistes2)
DetectorError = re.compile(r"'error':\sFalse")
ErrorValido = DetectorError.findall(texto)
if cant == len(ErrorValido):
    print("Basado en las expresiones regulares, se encontraron", cant, "'error: False' "
          "en la lista de chistes")
```

Entonces empezamos por crear una copia de la lista de diccionarios que resulta de la consulta a la API y lo guardamos en chistes2 luego lo que hacemos es que convertimos chistes2 a string y lo guardamos en la variable texto, esto lo hicimos ya que las expresiones regulares solo pueden buscar en datos tipo str, igual que en el pasado usamos re.compile y la expresión que pusimos fue la palabra 'error' con las comillas simples, los dos puntos :, \s para representar un espacio y la palabra False, prácticamente lo que la API da en su formato exacto, y lo mandamos a buscar las veces que aparezca en la cadena de caracteres texto con .findall (esperamos que la cantidad de veces que se encuentre esta expresión regular sea igual a la cantidad de chistes que se pidieron, para así saber que no hubo ningún error), luego hacemos el if que va a ejecutarse si lo antes mencionado es verdad y ya por último imprime ese comentario

-Para validar que el ID sea un número

```
#validar con expresiones regulares que el ID sea un numero
DetectorId = re.compile(r"\d+")
IdValido = DetectorId.findall(texto)
#print(IdValido)
if len(IdValido) == 0:
    print("Hubo un fallo en los ID")
else:
    print("No hubieron errores en los IDS, todos son numeros según las expresiones regulares")
```

Seguimos un proceso casi igual que en los dos anteriores, y reutilizamos la variable texto para que la expresión regular funcione, en este caso la expresión regular fue muy simple ya que solo usamos \d+ para que se busque un digito o más con el .findall, haciendo referencia a encontrar los IDS (El ID es el número de chiste) y validar que la API haya regresado un número y no algún otro dato que no coincide

En conclusión las herramientas que empleamos fue re.compile(), .findall(), texto (variable que guarda la lista de diccionarios convertida a str) y len()

- **Técnicas de limpieza aplicadas.**

  Consideramos que la API que usamos da poca información con la cual trabajar, por lo que concordamos en que no era conveniente quitarle información

- **Estructura de los datos optimizados y su diseño lógico**

Para la estructura de datos que da la API es una lista de diccionarios, entonces no quisimos guardarla en alguna estructura nueva, ya que consideramos que la lista de diccionarios es fácil de entender y de manipular sus datos, asimismo el acceso a ellos es práctico. Ahora, para guardar la información de las consultas básicas si quisimos organizarlas en una estructura de datos, decidimos una lista de diccionarios también

```
106 DatosProcesados = {
107     "chistes_una_parte": "",
108     "chistes_dos_partes": "",
109     "prom_chistes_una_linea": "",
110     "prom_chistes_dos_lineas": "",
111     "prom_exito_ejecucion": ""
112     }
113 
114 DatosProcesados["chistes_una_parte"] = contador_single
115 DatosProcesados["chistes_dos_partes"] = contador_twopart
116 DatosProcesados["prom_chistes_una_linea"] = promedio_single
117 DatosProcesados["prom_chistes_dos_lineas"] = promedio_twopart
118 DatosProcesados["prom_exito_ejecucion"] = prom_error
119 
120 
121 ListaDatosProcesados = []
122 ListaDatosProcesados.append(DatosProcesados)
123 
```

Llamamos al diccionario DatosProcesados y dentro de el creamos una clave para cada consulta básica que procesamos anteriormente y dejamos el valor vacío, luego llenamos el diccionario con sus variables respectivamente, después

creamos la lista vacía ListaDatosProcesados y por último usamos .append() y le agregamos el diccionario.

Con esto ya tenemos las consultas básicas ordenadas en una sola estructura de datos la cual es fácil visualizar, manipular y obtener acceso a datos específicos