Clase 3

Repaso

Bucle

for ... in ... : Los bucles o ciclos, son instrucciones que ejecutan repetidas veces un bloque de código. La sentencia **`for`** va a recorrer uno a uno todos los elementos de una lista y los va a asignar en una variable que se puede usar DENTRO de su bloque de código (fuera del **`for`** ya no estará definida esa variable). La variable de la que hablamos puede llevar cualquier nombre, y se coloca entre el **`for`** y el **`in`**. La lista que se recorre, se coloca entre el **`in`** y los dos puntos.

Repaso

Indentation

\t Este es un concepto que debe quedar muy claro. Recuerden que dependiendo de la sangría que le den a su código, es la forma de decirle a Python dónde comienzan y dónde terminan los bloques de código. Es importante contar bien 4 espacios para cada separación, o un tabulador, fíjense que el código que está en el mismo bloque quede a la misma altura, y que el que no, quede fuera, pegado al margen o a la altura que le corresponde. En el título de esta lámina escribí *indentation*, con ese nombre Python les marcará un error cuando detecte (o crea detectar) errores de sangría. El símbolo al inicio de este texto son los caracteres especiales que pueden usar si quieren que Python escriba tabuladores (como **\n** para salto de línea).

Repaso

Otras instrucciones

strip() Esta es una función de las variables de tipo texto. Sirve para eliminar el espacio al inicio y al final del texto, es decir lo "limpia"de los espacios blancos sobrantes.

pass Esta instrucción significa "no hagas nada". Podría parecer inútil, pero si quieren tener un bloque de código VACÍO, es necesario que tenga al menos esto, si lo dejan de verdad vacío va a marcar un error.

continue Esta instrucción es propia de los bucles, su función es que, llegado al punto en el que se encuentra, ignora el resto del bloque y pasa a la siguiente iteración del bucle.

break Esta instrucción es similar a `continue`. La diferencia radica en que no pasa a la siguiente iteración, en su lugar, se sale por completo del ciclo, rompe la secuencia y continúa con el programa.

Repaso... y un poco más

Otras instrucciones

`!=` Este operador es otro operador lógico (como <, o >), es decir, *regresa* un valor de verdadero o falso. Al contrario que el operador ==, este operador *regresa* VERDADERO cuando los elementos comparados son DIFERENTES.

`+=` Otro operador de asignación, similar a =. La diferencia está en que este utiliza el valor original de una variable para reasignarla, junto con un incremento, es decir:

```
variable+=1
```

es un atajo para:

```
variable=variable+1
```

Mucha gente encuentra esto mas claro, y cómodo. Además, funciona también para decrementos (-=)

Ejercicio 4

Para recordar y dejar claros los nuevos conceptos:

- Muestren en pantalla el contenido de un archivo desde el inicio hasta la primera línea vacía.

Para los que buscan un mayor reto:

- Armen una lista, en cada elemento coloquen todo el texto que tengan hasta que encuentren una línea vacía. Es decía, separen el texto por líneas vacías, no por saltos de línea.

Archivos ... en serie

Lista de archivos

No solo las oraciones se pueden manejar en listas:

```
import os

carpeta_nombre="Documentos\\"

archivos_lista=os.listdir(carpeta_nombre)

for archivo_nombre in archivos_lista:
        print(archivo_nombre)
```

- Aquí, hacemos uso de una nueva instrucción: el **`import`**
- Python tiene muchísimos complementos que tienen diferentes funciones, no los tiene precargados todos porque sería algo demasiado pesado. Es por eso que se utiliza el **`import`** cuando se quieren utilizar funciones que no están en el "paquete básico" de Python.

Lista de archivos

No solo las oraciones se pueden manejar en listas:

```
import os

carpeta_nombre="Documentos\\"

archivos_lista=os.listdir(carpeta_nombre)

for archivo_nombre in archivos_lista:
        print(archivo_nombre)
```

- En este caso, importamos el módulo **os**, que contiene funciones para interactuar con el sistema operativo.
- La función **listdir()** del módulo **os** permite obtener el contenido de una carpeta en forma de lista.

Lista de archivos

- Ahora podemos ejecutar comandos para todos los archivos de nuestra carpeta de trabajo.

```
import os

carpeta_nombre="Documentos\\"
archivos_lista=os.listdir(carpeta_nombre)

for archivo_nombre in archivos_lista:
    archivo=open(carpeta_nombre+archivo_nombre)
    lineas_lista = archivo.readlines()
    archivo.close()
    longitud = len(lineas_lista)
    print("El archivo",archivo_nombre,"tiene",longitud,"lineas")
```

Ejercicio 5

Muhcas veces se quiere analizar documentos relacionados como si fueran uno solo, ya sea que se traten documentos del mismo autor, mismo tema, o alguna característica común.
Nosotros tenemos documentos legales del Instituto Federal de Telecomunicaciones, el ejercicio consiste en:

- Obtener un documento (por ejemplo `"UNION.txt"`) que contenga la unión de todos los documentos del corpus.
  - Recuerden que pueden hacer iteraciones sobre sus documentos.
  - Utilicen una variable para ir acumulando el texto de los documentos.
  - Recuerden que pueden concatenar texto con el operador '+'

Segmentacion

Lista de palabras

- Ya que sabemos que podemos trabajar con nuestros archivos uno tras otro, volvamos a trabajar enfocándonos en uno.
- Esta vez, separaremos palabras.

```
carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

palabras_lista=texto.split()
print(palabras_lista)
```

Lista de palabras

- Por cierto, podemos ordenar una lista, con su función **sort()**.

```
carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

palabras_lista=texto.split()
palabras_lista.sort()
for palabra in palabras_lista:
        print(palabra)
```

Lista de palabras

- La función `split()` divide el texto según los espacios que encuentra.
- Esta es la forma mas simple de separar palabras.
- Pero seguramente notarán algunos detalles del proceso. Principalmente, notarán que muchas palabras de la lista también tienen símbolos de puntuación como comas, puntos y paréntesis que no forman parte de la palabra. Es necesario un método más detallado y confiable para la separación.
- En PLN, a este proceso de segmentar el texto se llama *tokenización*. Y a los elementos obtenidos (palabras, signos de puntuación, urls, siglas, fechas, etc.) se le llaman *tokens*.

Tokenización

¿Cómo hacer un tokenizador?

- Tenemos que ser capaces de separar las palabras de la puntuación con la que colindan.
- Sin embargo, hay que considerar que no toda la puntuación se debe separar, tal es el caso de los números, las fechas, las siglas, entre otros.

Tokenización

forma básica

- De nuestra lista de palabras, podemos observar que en la mayoría de los casos, lo único que obstruye a las palabras son símbolos como paréntesis, comas, puntos y punto y comas.
- Un tokenizador sumamente básico se lograría remplazando esos signos para separarlos del texto, y más adelante usar la función `split()`.

Tokenización

forma básica

```
carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

simbolos=["(",")",",",".",";",":","\""]

for simbolo in simbolos:
        texto=texto.replace(simbolo," " + simbolo + " ")

palabras_lista=texto.split()
palabras_lista.sort()
for palabra in palabras_lista:
        print(palabra)
```

Tokenización

- En el programa anterior vimos el uso de la función `replace()` de los textos, como una forma sencilla de separar símbolos.
- Sin embargo, podemos ver en la lista de palabras resultantes, que ahora se están separando cosas que sería mejor mantener unidas, como las siglas.
- Para lograr esto, es necesario una serie de reglas del estilo:
  - Si el símbolo está entre dos **letras mayúsculas**, no lo quites.
  - Si el símbolo está entre dos **números**, no lo quites.
- Pero cómo decirle a la computadora qué es una **letra mayúscula** o qué es un **número**.
- Existe una herramienta muy poderosa que es capaz de manejar el texto con grupos de letras (como mayúsculas, minúsculas, digitos, etc) y reglas de repetición.