Clase 2

Intérprete El intérprete ES python. Es un programa que tiene una lista casi interminable de instrucciones que es capaz de reconocer.
Nosotros vimos funciones como `print()` y `open()`, o comandos como `with` y `del`, cada uno tiene un significado para la computadora, así como cuando en Word damos la instrucción de centrar el texto

Intérprete No se espera que uno memorice todas las funciones que tiene el lenguaje de programación, para eso existe google. Lo que SI se espera, es que obtengan una noción de los alcances que tiene una función. Es cierto, uno como programador puede hacer programas tan complejos como desee, pero la manera de lograrlo es tras utilizar en conjunto las pequeñas herramientas de las que dispone (o algunas no tan pequeñas).

Intérprete En este curso, para facilitar instalación y asegurarse de que se tenían todos los paquetes necesarios se recomendó instalar Anaconda.
Pero eso no quiere decir que se NECESARIO, Anaconda incluye Python, pero Python no es exclusivo de Anaconda. Los usuarios de Linux encontrarán que tienen Python en el sistema por defecto.

Editor En cuanto al editor, CUALQUIER editor funciona. Sepan que la recomendación de Spyder es equivalente a la recomendación de Anaconda. Vienen en paquete. En este curso no hay preferencia por Spyder, simplemente es más fácil instalar una sola cosa que dos. Si se sienten más cómodos con otro editor, en el curso no habrá nada exclusivo de Spyder (ni de Anaconda para el caso).

Editor Otra cosa muy importante, el editor no tiene nada de mágico, eso es texto. Notarán que si toman el texto y lo introducen directamente en el intérprete lograrán el mismo efecto que cuando corren desde el editor. El editor solo muestra con colores instrucciones que reconoce. Al momento de ejecutar el programa, manda al intérprete línea a línea lo que han escrito. Eso es la programación, dar a una computadora una serie de instrucciones en orden para que las ejecute.

Todo lo que Python vea después de un signo de gato (hasta el fin de la línea) será ignorado. Esto es muy útil, por ejemplo, para quitar momentáneamente código que no queremos perder. Pero el mayor beneficio que se tiene es la información que se puede dar a los lectores del programa, sean otras personas, o uno mismo en el futuro. Pueden usarlos como si fueran apuntes.

Repaso

Variables y funciones

Variables Una variable es una caja en la que puedes guardar el resultado de casi cualquier operación (las operaciones que escriben en pantalla no se guardan, como `print()`). Usen nombres de variables que dejen claro para qué las están usando.

Funciones Las funciones se distinguen porque llevan paréntesis después de su nombre, son instrucciones que Python reconoce y ejecuta. Las funciones por lo regular tienen como resultado un valor que se puede (y casi siempre se debe) guardar en una variable, a esto se le llama que *regresan* el valor.

Repaso

Funciones

print() Muestra en pantalla lo que recibe entre paréntesis. Si es texto (es decir, si se da entre comillas) lo muestra tal cuál, si no, primero lo evalúa (eso incluye sustituir variables por su valor) y luego lo muestra.

open() Recibe la ubicación de un archivo (puede ser relativa), y lo abre. Esta función *regresa* un archivo abierto, eso se escucha raro, pero pueden verlo como que lo prepara para su lectura, es decir, les da a ustedes un objeto (una variable) que tiene la función de darles el texto `read()`. Más claramente para algunos, esta función toma un archivo del disco duro y lo pasa a la memoria RAM para que puedan leerlo (eso mismo pasa cuando abren un pdf por ejemplo).

close() Cerrar los archivos después de usarlos es buena práctica. Esta función pertenece a los archivos *regresados* por la función **`open()`**.

Es importante recordar que cuando una función *pertenece* a una variable (o mejor dicho a un tipo de variable), la forma de usarlas, es a través de la variable a la que pertenecen. Se usa el nombre de la variable, punto, el nombre de la función.

```
variable.funcion()
```

Repaso

Funciones

read() La función `open()` *regresa* un archivo abierto, este archivo lo deben conservar en una variable, de esa manera pueden usar esta función. Ésta función pertenece a los archivos *regresados* por la función `open()`. A su vez, esta función *regresa* el texto contenido en el archivo que abrieron.

write() Muy similar a la función anterior pero al revés. La función `open()` *regresa* un archivo abierto, si se usa la opción `"w"` se abrirá en modo escritura y se puede usar esta función. `write()` escribe el contenido de los paréntesis en un archivo. Recuerden que NO incluye salto de línea (como `print()`.

Repaso

Otras instrucciones

with ... as ... : Esta instrucción abre un bloque de código (recuerden, esos siempre comienzan con 4 espacios o tabulador) en el que una variable (que va después del **`as`** está definida, y SÓLO en ese bloque. Esto se usa mucho para abrir documentos, sacar su contenido y que se cierren solos después del bloque de código.

in Esta es una instrucción lógica, eso quiere decir que *regresa* verdadero o falso (útil para los `if`. revisa si un texto se encuentra dentro de otro. Esta operación no se efectúa para cada una de las apariciones, es o si o no.

\ Este es un símbolo especial, da un significado extra a la letra que le sigue (no todas las letras aplican). Vimos el símbolo `\n`, que quiere decir salto de línea. El símbolo `\"`, con el que se pueden usar comillas dentro del texto (que está entrecomillado y unas comillas sin diagonal invertida lo cortarían). El símbolo `\\`, con el que podemos escribir la diagonal invertida, ya que, siendo un símbolo especial, no se toma como símbolo normal al ser usado solo.

Repaso

Condicionales

if ... : Evalúa una función lógica (de esas que *regresan* un valor de verdadero o falso, como `<`, `>`, `==` o `in`, entre otras). Si la condición se cumple, ejecuta un bloque de código, si no, se lo salta.

elif ... : Esta instrucción se puede usar después de un `if`, si la condición del `if` falla, aquí se puede probar una nueva condición, su comportamiento entonces es el mismo que el del `if`.

else : Esta instrucción se puede usar después de un `if` y una posible cadena de `elif`'s. El bloque del `else` se ejecuta si todas las otras condiciones fallan.

Repaso... y un poco más

Documentos

Texto La mejor forma de trabajar con el contenido de documentos de forma automática, es si los documentos tienen texto plano, un archivo TXT, puro contenido. Puede que ustedes se topen con colecciones de textos (corpus) que están en pdf, o word. Hay formas de convertir dichos documentos a texto, pero no cubriremos esa parte en este curso, en su lugar, me tomaré la libertad de recomendar GECO, es un gestor de corpus en línea `www.corpus.unam.mx/geco` en el que pueden subir sus documentos en estos formatos y los convertirá por ustedes (entre otras cosas).

Repaso... y un poco más

Documentos

Codificación Algo que SI nos compete y que no fue discutido en la clase pasada, es la codificación, la codificación es la forma en la que se representa cada letra y símbolo internamente en la computadora (unos y ceros). Probablemente han escuchado hablar del UTF-8. Es una codificación que incluye letras y símbolo: árabe, braille, copto, cirílico, griego, sinogramas (hanja coreano, hanzi chino y kanji japonés), silabarios japoneses (hiragana y katakana), hebreo, latino, cuneiforme, griego antiguo, lineal B micénico, fenicio, rúnico, símbolos musicales y matemáticos, fichas de juegos de dominó, flechas, iconos etc.

Repaso... y un poco más

Documentos

Codificación UTF-8 es bastante estándar (89.2 % del internet, y eso incluye a GECO). Tristemente, Windows y Mac no leen con codificación UTF-8 por defecto (Linux si) y por lo tanto Python tampoco en ambos sistemas (en Linux si).
Inicialmente teníamos un corpus para cada sistema operativo. A partir de hoy en la página encontrarán solo uno, con UTF-8 como codificación (si, es el que originalmente era para Linux).

Repaso... y un poco más

Documentos

Codificación ¿Cuál será la diferencia? La diferencia será que se tendrá que especificar explícitamente a Python, al momento de abrir un archivo, que lo haga usando la codificación UTF-8

```
archivo_abierto=open("C:\\RUTA\\Texto.txt","r")
```

pasa a:

```
archivo_abierto=open("C:\\RUTA\\Texto.txt","r",encoding="utf8")
```

Repaso... y un poco más

Documentos

Rutas La clase pasada vimos rutas absolutas, es decir, rutas completas desde el disco local (C: o la raíz /). Pero también están las rutas relativas. Las rutas relativas se dan dependiendo de el lugar en el que se guarde el archivo en el que están programando (de allí que se llamen relativas).

Repaso... y un poco más

Documentos

Rutas Con su archivo guardado (algo que no había mencionado pero que es importante, es que la extensión del archivo python debe ser `.py`) en una carpeta, si en esa misma carpeta se encuentra la carpeta de **`Documentos`** que hemos estado manejando.

```
"C:\\Users\\user\\...\\Documentos\\Texto.txt"
```

pasa a:

```
"Documentos\\Texto.txt"
```

ÚNICAMENTE aplica si el archivo de programación (el `.py`) está en la misma carpeta en la que está la carpeta de **`Documentos`** (en este caso, o en la ruta que quieran leer).

Repaso

Listas

- Algo interesante, las variables pueden tener más de un valor.
- Una variable puede ser una lista.

```
semana_laboral=["Lunes","Martes","Miércoles","Jueves","Viernes"]
print("Semana laboral =",semana_laboral)
print("Dia 1 =", semana_laboral[0])
print("Dia 2 =",semana_laboral[1])
print("Dia 3 =",semana_laboral[2])
print("Dia 4 =",semana_laboral[3])
print("Dia 5 =",semana_laboral[4])
```

- Las listas pueden agrupar un conjunto de datos relacionados entre sí.
- IMPORTANTE: los índices de las listas, comienzan en 0.

Listas

- Si utilizamos los índices de la lista, los valores que contiene se pueden manipular individualmente como cualquier otra variable.

```
semana_laboral[4]="Sabado?"
print("Semana laboral cambiada =",semana_laboral)
```

Listas

- Las listas tienen una serie de comandos propios que es muy útil tener en cuenta:

```
longitud_lista=len(semana_laboral)
print("Tamaño de la lista =",longitud_lista)

num_elemento=semana_laboral.index("Martes")
print("Lugar del Martes =",num_elemento)

semana_laboral.append("Viernes")
print("Semana laboral (append(\"Viernes\")) =",semana_laboral)

del semana_laboral[4]
print("Semana laboral (sin el lugar 4)) =",semana_laboral)

semana_laboral.remove("Viernes")
print("Semana laboral (sin \"Viernes\")) =",semana_laboral)
```

Listas

- Y ahora que ya sabemos de listas, hay que utilizarlas.
- Una forma muy intuitiva de manejar archivos, es por líneas. Python sabe, por supuesto, leer un archivo por líneas y ponerlas en una lista:

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="DOF_P_IFT_290216_71_Datos_Relevantes_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        lineas_lista=archivo.readlines()

print(lineas_lista)
```

- Muchos documentos tienen cierta estructura e información separada en líneas, como las listas o las tablas.

Bucles

'for'

- Los bucles son los procesos iterativos.
- La misma acción se hace una y otra vez para un conjunto de datos, por lo general para todos los elementos de una lista.
- Por ejemplo, podemos hacer que para cada línea de un archivo Python la escriba en la pantalla:

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\Legales\\"
archivo_nombre="DOF_P_IFT_290216_71_Datos_Relevantes_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        lineas_lista=archivo.readlines()

for linea in lineas_lista:
        print(linea)
        print()
```

Bucles

'for'

- Veamos un ejemplo un poco mas complejo:

```
01: carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
02: archivo_nombre="DOF_P_IFT_291116_672_Acc.txt"
03:
04: with open(carpeta_nombre+archivo_nombre,"r") as archivo:
05:         lineas_lista=archivo.readlines()
06:
07: num_linea=1
08: for linea in lineas_lista:
09:         print("LINEA",num_linea,":",linea)
10:         num_linea=num_linea+1
11:
12: print("FIN DE ARCHIVO")
```

Bucles

'for'

- Los bloques de código se pueden anidar.

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="DOF_P_IFT_291116_672_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        lineas_lista=archivo.readlines()

num_linea=1
for linea in lineas_lista:
        if linea == "":
                pass
        else:
                print("LINEA",num_linea,":",linea)
                num_linea=num_linea+1

print("FIN DE ARCHIVO")
```

Bucles

'for'

- El código anterior "falla" para eliminar las líneas vacías, probemos este otro:

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="DOF_P_IFT_291116_672_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        lineas_lista=archivo.readlines()

num_linea=1
for linea in lineas_lista:
        if linea.strip() == "":
                continue
        print("LINEA",num_linea,":",linea)
        num_linea=num_linea+1

print("FIN DE ARCHIVO")
```

Ejercicio

Por orden de dificultad, modifiquen el programa para que:

- Cuente las líneas que contiene el archivo y lo indique.
- Cuando encuentre una línea que NO contenga texto, indique que esa línea esta vacía.
- Muestre cuántas líneas tienen texto y cuántas no.