

# PLN con Python

Alejandro Pimentel

## Clase 1

# Objetivos del PLN

- ▶ Crear aplicaciones que puedan manipular, interpretar y generar lenguaje humano
- ▶ Modelar la capacidad lingüística humana.
- ▶ Representar el conocimiento lingüístico de una manera computacional.

- ▶ Introducción al lenguaje de programación Python.
- ▶ Instrucción a la programación y módulos con un enfoque para PLN.
  - ▶ Expresiones regulares
  - ▶ NLTK (natural language tool kit).

- ▶ Intérprete Python
- ▶ Editor de texto
- ▶ Internet



ANACONDA®

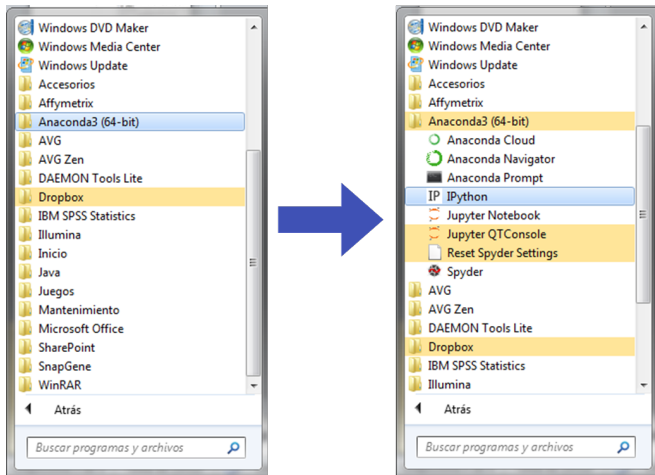


spyder

<https://www.continuum.io/downloads>

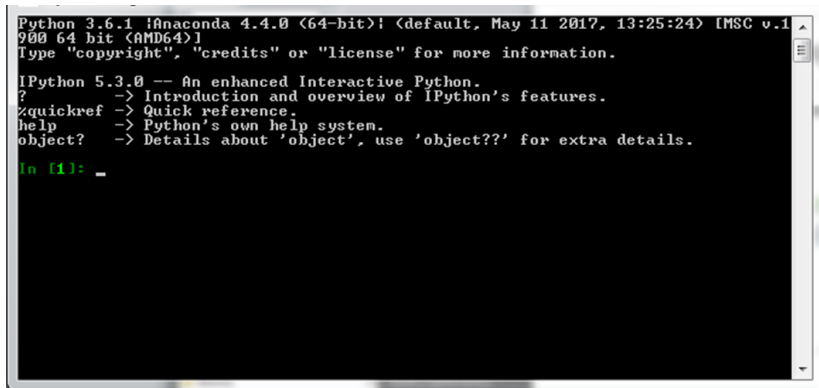
# Primero lo primero

Depende de la forma en la que hayan instalado python, pueden comenzar a usarlo de diferentes maneras, como por la línea de comandos (cmd). Si instalaron anaconda, podrán entrar desde allí.



# Python en consola

- ▶ Esta pantalla es la consola de python.



```
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]: _
```

- ▶ Aquí podrán escribir los comandos directamente para que python los interprete y ejecute.



# Hola Mundo

en Python

- ▶ El primer programa por excelencia, el "Hola Mundo"
- ▶ La consola recibe la instrucción de mostrar en la pantalla un mensaje.

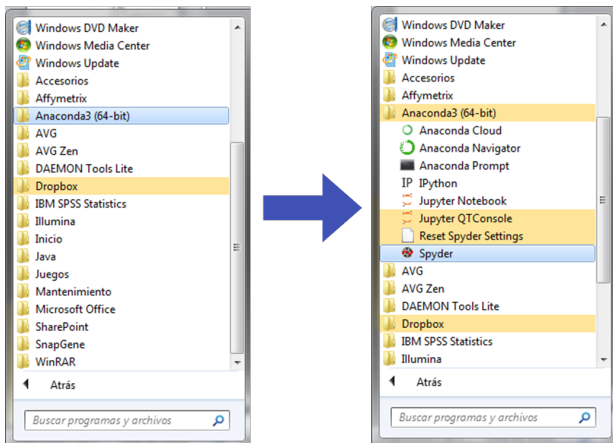
```
print("Hola Mundo")
```

- ▶ El mensaje debe ir entre comillas.

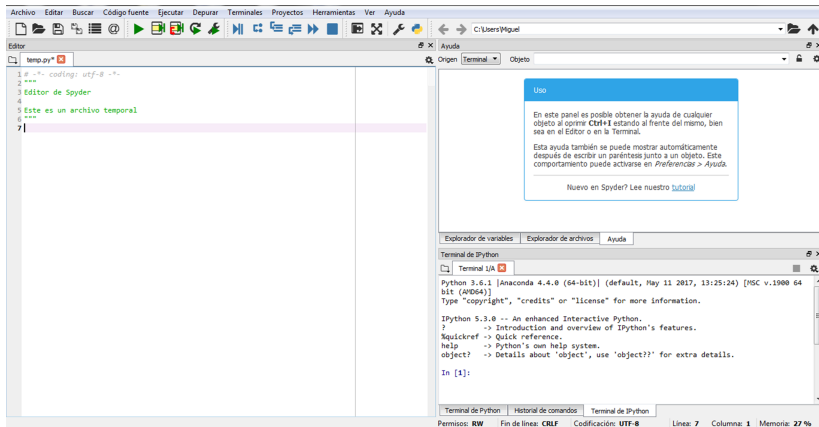
# El editor

Python, al igual que la mayoría de los lenguajes de programación, no necesita nada más complejo que un bloc de notas. Pero existen un sinnúmero de opciones, pueden usar la que más les guste.

Si instalaron anaconda, y no tienen un editor preferido, pueden probar *Spyder*.



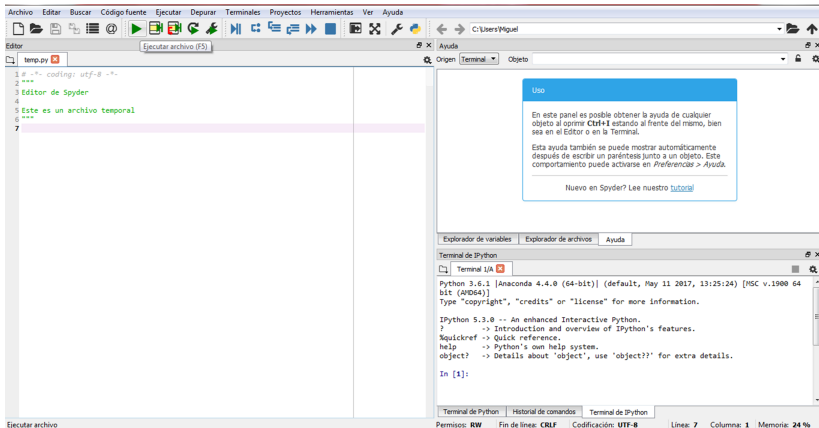
- ▶ Spyder cuenta con una consola en la que se pueden ejecutar comandos, o lo que se esté escribiendo en el editor.
- ▶ Además, cuenta con un apartado de ayuda para las funciones de Python.



- ▶ Dentro del editor podemos escribir una serie de comandos que queremos que se ejecuten.
- ▶ Serán leídos e interpretados en orden, uno tras otro.

```
print("De ola en ola,")  
print("de rama en rama,")  
print("el viento silba")  
print("cada mañana.")
```

- ▶ Una vez que se escriben las instrucciones, se puede ejecutar el archivo completo.
- ▶ Varios editores cuentan con esta función.



# Operaciones

- ▶ Python también es capaz de hacer operaciones matemáticas, cual si fuera una calculadora simple.

```
print(2 + 2)
print(3 * 4)
print(100 - 1)
```

- ▶ Es importante notar que en este caso, no se usan comillas en el interior de la función `print()`.

- La razón, es que las comillas se utilizan cuando se está manejando texto, los números, para hacer operaciones, se deben tomar como números.

```
print("2" + "2")  
print("3 * 4")  
print("100" - 1)
```

# Operaciones

- ▶ Ambas formas se pueden combinar dentro de la función `print()`.

```
print("2 + 2 =", 2 + 2)
print("3 * 4 =", 3 * 4)
print("100 - 1 =", 100 - 1)
```

- ▶ Es importante notar que el texto y los números están separados por una coma.



Entonces se tienen:

- ▶ Palabras, el texto que se debe manejar entre comillas.
- ▶ Números y operaciones, que deben ir sin comillas para que se evalúen.

¿Qué hay de las palabras sin comillas?

```
print(Hola)
```

# Variables

- ▶ Las variables son palabras (o cualquier combinación de letras en realidad) que se utilizan para guardar valores.
- ▶ Cualquier valor o el resultado de cualquier operación puede ser asignado a una variable.
- ▶ Los valores se asignan con el símbolo =

```
nombre = "Juan Hernández Hernández"  
suma = 2 + 3  
print("Hola" , nombre)  
print("2 + 3 =", suma)
```

# Variables

- ▶ Las variables también pueden ser utilizadas para asignar otras variables.
- ▶ O incluso para cambiar su propio valor.

```
x = 2 + 3
y = x + 5
print("x =", x)
print("y =", y)

y = y + 10
print("y =", y)
```

## IMPORTANTE

Dejen que su código les hable.

Y háganlo aunque les de flojera.

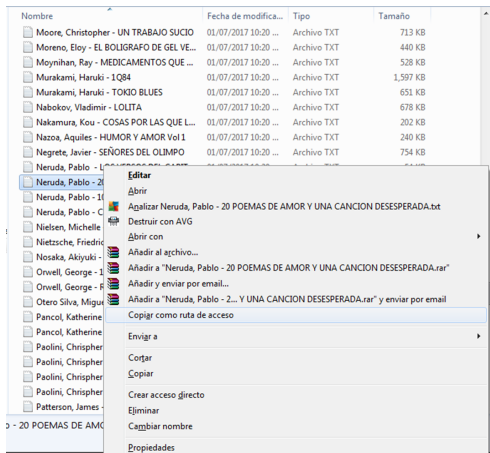
```
x = 2 + 3 # A x se le asigna el resultado de una suma.  
y = x + 5 # Se puede usar una variable asignada para hacer  
          # operaciones.  
print("x =", x) # El comando print() sirve para  
                # mostrar cosas en pantalla.  
print("x + 5 =", y) # Puede recibir texto o números,  
                   # incluso operaciones, pero separados por  
                   # comas.
```

Archivos

- ▶ Un corpus es un conjunto de documentos (texto, video, audio, etc.) destinado a la investigación.
- ▶ La recolección de un corpus depende de lo que se busca estudiar.
- ▶ Para este curso, contamos con un conjunto de documentos legales, sin embargo, si cuentan con un corpus propio pueden utilizarlo.

# Ruta de archivos

- ▶ Es común necesitar la ruta de los archivos.
- ▶ En Windows, se puede usar **SHIFT+CLICK\_DERECHO** en un archivo, lo que despliega un menú de opciones en el que aparece: **"Copiar como ruta de acceso"**



- Para comenzar a trabajar con un archivo, lo primero que se tiene que hacer es abrirlo.

**IMPORTANTE** Las rutas de los archivos en Windows separan las carpetas con diagonal invertida ('\\'). Éste es un símbolo especial que no se puede usar así nada mas, más adelante veremos por qué. Por ahora solo recuerden que deben duplicar la diagonal invertida ('\\' en lugar de '\\').

```
archivo_abierto=open("C:\\\\RUTA\\P_IPT_290216_73_Acc.txt")
```



- ▶ Ya que el archivo está abierto podemos leer el contenido.

```
texto=archivo_abierto.read()
```

- ▶ `read()` es una función de los archivos (debe ser un archivo que haya sido abierto con la función `open()`) que lee el texto y lo "regresa". Esto quiere decir que el resultado puede (o mejor dicho **debe**) asignarse a una variable, en este caso, la variable `texto`
- ▶ Es una función que no recibe ningún argumento, por eso los paréntesis vacíos.

- ▶ Podemos hacer que Python nos muestre el contenido del archivo, que ahora está en la variable `texto`.

```
print(texto)
```

- ▶ Recuerden que en esta ocasión NO se usan comillas, queremos usar la variable `texto` , no el texto: *"texto"*.
- ▶ Finalmente, es importante cerrar el archivo que abrimos en primer lugar:

```
archivo_abierto.close()
```

- ▶ El programa completo se vería algo así:

```
archivo_abierto=open("C:\\\\RUTA\\\\P_IFT_290216_73_Acc.txt" )
texto=archivo_abierto.read()
print(texto)
archivo_abierto.close()
```

- ▶ Más sus respectivos comentarios

# Escritura de archivos

- ▶ El proceso de escritura es similar, pero se distingue desde la forma en la que se abre el archivo

```
archivo_abierto=open("C:\\\\RUTA\\mi_archivo_nuevo.txt","w")
```

- ▶ Noten dos cosas.
- ▶ Primero, la 'w' como segundo argumento de la función `open()`. Esa 'w' significa *write*, y se usa para abrir el archivo en modo escritura (si se omite, como en el caso anterior, por defecto los archivos se abren en modo lectura: `r`).
- ▶ Segundo, el nombre del archivo. NO hagan esto con un archivo existente. Cuando se abre un archivo en modo escritura, se crea como nuevo y Python no pregunta si quieres sobrescribir, lo hace.

# Escritura de archivos

- ▶ Para escribir en el archivo, usaremos la función `write()`.

```
archivo_abierto=open("C:\\\\RUTA\\mi_archivo_nuevo.txt","w")  
archivo_abierto.write("Esto se escribe en el archivo")  
archivo_abierto.write("Esto tambien")
```

- ▶ La función `write()` es muy parecida a `print()`, con la diferencia de que manda el texto al archivo en lugar de a la pantalla.
- ▶ Para ver los cambios en su archivo, no olviden cerrarlo.

```
archivo_abierto.close()
```

# Escritura de archivos

- ▶ Si abren su archivo verán lo que escribieron, y lo más probable es que se topen con algo como esto.

```
Esto se escribe en el archivoEsto tambien
```

- ▶ Probablemente este no era el resultado que esperaban.
- ▶ Una diferencia entre `print()` y `write()` es que `write()` no añade un salto de línea en el archivo como `print()` en la pantalla.

# Escritura de archivos

- ▶ Para agregar saltos de línea a voluntad se usa el símbolo: `\n`
- ▶ Por lo tanto, el programa anterior, sería mejor escribirlo algo así:

```
archivo_abierto=open("C:\\\\RUTA\\mi_archivo_nuevo.txt","w")

archivo_abierto.write("Esto se escribe en el archivo\n")
archivo_abierto.write("Esto tambien\n")
archivo_abierto.write("Mira, puedo escribir \"comillas\"\\n")
archivo_abierto.write("Gracias a la diagonal invertida: \\ \n")

archivo_abierto.close()
```

- ▶ Al agregar un `\n` al final de cada texto en el que se use la función `write()`, se obtiene el mismo comportamiento que con `print()`.
- ▶ Como ven, la diagonal invertida es un símbolo especial. También se puede usar para escribir comillas dentro de los mensajes y que no se malinterpreten como el cierre de comillas.

# Ejemplo

```
c="C:\\Users\\user\\Desktop\\Documentos\\"
e="P_IFT_290216_73_Acc.txt"
s="archivo_nuevo.txt"
e2=open(c+e,"r")
s2=open(c+s,"w")
t=e2.read()
t2=t
s2.write(t2)
e2.close()
s2.close()
```



# Ejemplo

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"
salida_nombre="archivo_nuevo.txt"

entrada_abierto=open(carpeta_nombre+archivo_nombre,"r")
salida_abierto=open(carpeta_nombre+salida_nombre,"w")

texto_entrada=entrada_abierto.read()
texto_salida=texto_entrada

salida_abierto.write(texto_salida)

entrada_abierto.close()
salida_abierto.close()
```

# Manejo de archivos

- ▶ Aquí presentaré un código con otra manera de trabajar con un archivo abierto.
- ▶ Esta forma tiene la ventaja de ser mas clara.
- ▶ Y también más fácil ya que maneja por si misma el cierre del archivo.

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

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

# El programa continúa acá...
```

# Programación

# El 'if'

- ▶ El `if` es una instrucción que ejecuta un bloque si una condición se cumple.
- ▶ Es la instrucción más simple de control.

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="P_IFT_200416_162_Acc.txt"
palabra="acuerdo"

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

if palabra in texto:
    print("Encontré la palabra!")
if 2 > 5 :
    print("Dos es mayor que cinco!?")
```

## if ... else

- ▶ Ya que estamos viendo cómo funciona el `if` veamos también una instrucción que trabaja en conjunto, el `else`

```
carpeta_nombre="C:\\Users\\user\\Desktop\\Documentos\\"
archivo_nombre="P_IFT_200416_162_Acc.txt"
palabra="ACUERDO"

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

if palabra in texto:
    print("Encontré la palabra!!")
else:
    print("No encontré la palabra :(")
```

- ▶ Si la condición se cumple, se ejecuta el bloque del `if` y no el del `else`. Si la condición no se cumple, no se ejecuta el del `if` pero sí el del `else`

## if ... elif ... else

- Y para completar, la instrucción: **elif**

```
palabra1="ACUERDO"
palabra2="RESOLUCIÓN"

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

if palabra1 in texto:
    print("Encontré la palabra 1!!")
elif palabra2 in texto:
    print("No está la palabra 1 pero si la 2")
else:
    print("No hay ninguna de tus palabras :(")
```

- **elif** es la combinación de un **else** con un **if**, se usa como **else**, y es ignorado si el primer **if** se cumple, pero a su vez, también revisa la verdad de una condición para ejecutar su bloque de código.

# Ejercicio 1

- ▶ Modifiquen el ejemplo anterior para que el programa les indique si el documento es un acuerdo, una resolución o si es otra cosa.
  - ▶ Utilicen la variable. Es decir, si intercambian el contenido de `palabra1` y `palabra2` su programa debe seguir indicando correctamente qué tipo de documento es.

# 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.



- ▶ 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)
```