

**title:** Presentacion Python 2019  
**Author:** Claudia Banchoff, Viviana Harari  
**description:** clase 7  
**keywords:** Listas de comprensión, Manejo de excepciones y Avanzando en PySimpleGUI  
**css:** estilo.css

---

## **Seminario de Lenguajes - Python**

**Cursada 2019**

---

### **Temario**

- Repaso de la clase anterior.
  - Listas por comprensión
  - Manejo de excepciones.
  - Avanzando en PySimpleGUI
- 

### **Repaso. Por Python Plus!!**

---

### **Repaso. Por Python Plus!!**

- ¿Qué diferencias y semejanzas hay entre un archivo y un diccionario?
- ¿Qué es un archivo en Python?

open(nombre, modo)

↓  
Nombre del  
archivo

↓  
Forma de apertura:  
"r": read  
"w": write  
"a": append  
"x": create  
Opcionalmente: "+" y "b"

**Importante!!!** archivos  
binarios vs texto

height: 400px

- ¿Qué diferencias hay entre los modos "w", "x" y "a"?
- ¿Qué diferencias hay entre un archivo de texto y uno binario?

## Repaso. Por Python Plus!!

```
archi = open('archivo.txt')  
archil = open('otro.txt', "rb+")
```

- ¿De qué modo se abre estos archivos?
- ¿Cuándo pueden dar un error estas sentencias?
- ¿De qué formas puedo leer y guardar datos en un archivo?
- ¿Cómo detectamos el fin de archivo?

## Repaso. Por Python Plus!!

- ¿Qué está mal en el siguiente código?

```
import pickle  
import time  
  
archivo = open("superHeroesEnPICKLE.txt", "w")
```

```

datos = [
    {'nombre': 'Tony Stark', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", t
{'nombre': 'Bruce Wayne', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmt
    ]

pickle.dump(datos, archivo)

archivo.close()

```

- ¿Qué diferencias hay entre el módulo json y pickle?

## Repaso. Por Python Plus!!

- ¿Qué son stdin, stdout, stderr?
- ¿Cómo los abro?

## Listas por comprensión

- Las listas por comprensión son otra manera de crear listas
- Permite hacerlo de una manera más concisa. En vez de detallar todos sus elementos se coloca una **expresión** seguida de una **cláusula for** que representan a los elementos  
[ expresion for item in lista ]
- Hasta ahora si queríamos crear una lista, por ejemplo con los cuadrados de los primeros 100 números hacíamos:

```

cuadrados = []
for nro in range (100):
    cuadrados.append(nro**2)

```

- ¿Cómo lo haríamos en forma más concisa?

**class:** destacado

## Listas por comprensión

cuadrados = [nro\*\*2 for nro in range(100)]

## Listas por comprensión con expresiones condicionales

- Esta posibilidad nos permite quedarnos con ciertos elementos y descartar los restantes.
  - Para esto se debe utilizar una **cláusula if** después de la cláusula for.
  - Ejemplo: ¿Cómo haríamos si queremos listar los cuadrados pero SOLO de los primeros números pares?
- 

**class:** destacado

## Listas por comprensión con expresiones condicionales

cuadrados = [nro\*\*2 for nro in range(100) if nro % 2 == 0]

---

## Manejo de excepciones

- ¿Qué es un excepción?
  - ¿Qué hay que investigar del lenguaje cuando tiene manejo de excepciones?
  - Excepciones en Python
- 

## Analicemos el siguiente código

¿Qué puede suceder cuando lo ejecutamos?

```
alum_cod_libros_prestados = {33444555:['a33','a45'], 341122555:['b133'], 40999988:['a22']}

dni_alumno = int(input('Ingresa tu dni'))
while dni_alumno !=0:
    cod_libro = input('Ingresa cod_libro a pedir')
    alum_cod_libros_prestados[dni_alumno].append(cod_libro)
    dni_alumno = int(input('Ingresa tu dni'))
```

[Veamos](#)

¿Qué sucedió?

---

## Levantamiento de excepción

Se levantó una excepción al introducir un DNI que no coincidía con ninguna clave del diccionario

```
pruebaExc0.py
1 alum_cod_libros_prestados={33444555:['a33','a45'],34112255:['b133'],40999988:['a22','a45','b77']}
2
3 dni_alumno=int(input('Ingresa tu dni'))
4 while dni_alumno !=0:
5     cod_libro=input('Ingresa cod libro a pedir:')
6     alum_cod_libros_prestados[dni_alumno].append(cod_libro)
7     dni_alumno=int(input('Ingresa tu dni'))
8
9
10
11
12
13
14
15
16
17
```

```
C:\Windows\system32\cmd.exe
Ingresa tu dni33444555
Ingresa cod libro a pedirb67
Ingresa tu dni13423854
Ingresa cod libro a pedirn77
Traceback (most recent call last):
  File "pruebaExc0.py", line 6, in <module>
    alum_cod_libros_prestados[dni_alumno].append(cod_libro)
KeyError: 13423854
Presione una tecla para continuar . . .
```

class: destacado

## ¿Qué es una excepción?

Una **excepción** es un acontecimiento, que ocurre durante la ejecución de un programa, que **interrumpe** el flujo normal de las instrucciones de programa.

## Ejemplo de excepciones:

- Abrir un archivo que no existe.
- Acceder a una clave de diccionario que no existe .
- Buscar en una lista un valor que no existe.
- Invocar a un método que no existe.
- Referirse a una variable que no existe.
- Mezclar tipos de datos sin convertirlos previamente.
- Etc.

## Ejemplo de excepciones "sin manejar o gestionar"

```
>>> dic = {'juan':4,'pedro':5,'helen':6}
>>> dic[1]
'juan'
>>> dic[2]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    dic[2]
KeyError: 2
```

Ej. de excepción por **no existir la clave** solicitada. **KeyError**: nombre de la excepción levantada.

```
>>> print(m)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print m
NameError: name 'm' is not defined
```

Ej. de excepción por **variable no definida**. **NameError**: nombre de la excepción levantada.

```
>>> open('pepe.txt','r')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    open('pepe.txt','r')
IOError: [Errno 2] No such file or directory: 'pepe.txt'
```

Ej. de excepción por **no existir un archivo**. **IOError**: nombre de la excepción levantada.

# Python ¿maneja excepciones?

Hay lenguajes que:

- No presentan mecanismos para el manejo de excepciones. En esos casos podrían simular el manejo de las excepciones con otros recursos. Ej. Pascal
- Presentan mecanismos para el manejo de excepciones y, cuentan con recursos específicos para hacerlo. Ej.: Ada, Java, Ruby, etc.

## ¿Python?

---

## ¿Qué investigamos del lenguaje cuando tiene manejo de excepciones?

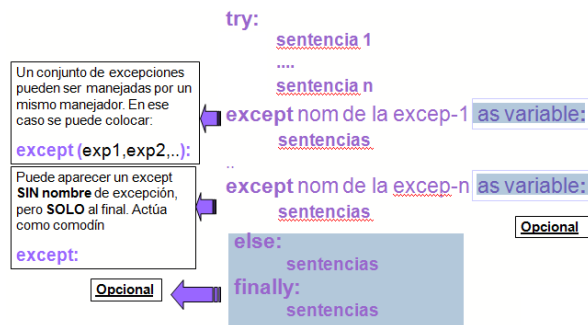
- ¿Qué acción se toma después de levantada y manejada una excepción?. ¿Se continúa con la ejecución de la unidad que lo provocó o se termina?
  - ¿Cuál es su ámbito?
  - ¿Cómo se alcanza una excepción?
  - ¿Cómo especificar la unidades (manejadores de excepciones) que se han de ejecutar cuando se alcanza las excepciones?
  - ¿Qué sucede cuando no se encuentra un manejador para una excepción levantada?
  - ¿El lenguaje tiene excepciones predefinidas?
- 

## Manejo de excepciones en Python

```
try:
    sentencia 1
    ....
    sentencia n
except nom de la excep-1:
    sentencias
..
except nom de la excep-n:
    sentencias
```

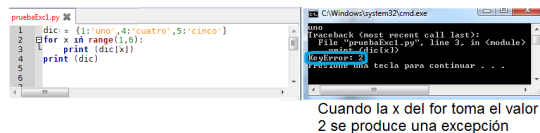
---

## ¿Estructura opcional?

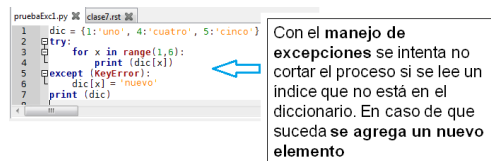


## ¿Usamos manejo de excepciones?

Si no utilizamos manejo de excepciones...



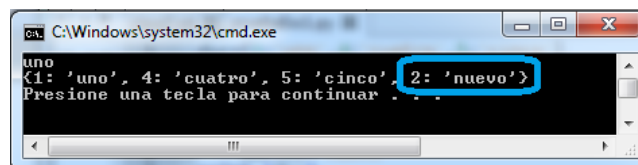
Si utilizamos manejo de excepciones...



Hagamos la prueba

¿Qué sucedió?

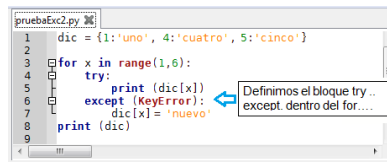
## Con manejo de excepciones...



Evitamos que el proceso se corte agregando el elemento al diccionario

Y ¿cómo hacemos para que el for siga hasta el final?..

## Si quisieramos que continúe el for...



```
1 dic = {'1': 'uno', 4: 'cuatro', 5: 'cinco'}
2
3 for x in range(1,6):
4     try:
5         print (dic[x])
6     except (KeyError):
7         dic[x] = 'nuevo'
8     print (dic)
9
```

Probemos

¿Qué sucedió?

---

## ¿Y las cláusulas else y finally?

- **else:**

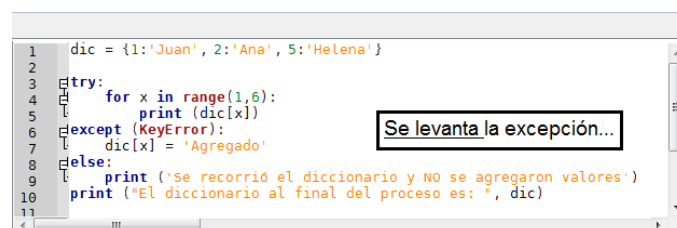
El código colocado en la cláusula else se ejecuta **solo si** no se levanta una excepción

- **finally:**

El código colocado en la cláusula finally se ejecuta **siempre** se haya o no levantado una excepción

---

## Cláusulas else

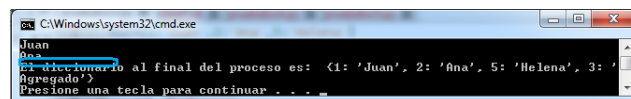


```
1 dic = {'1': 'Juan', 2: 'Ana', 5: 'Helena'}
2
3 try:
4     for x in range(1,6):
5         print (dic[x])
6     except (KeyError):
7         dic[x] = 'Agregado'
8     else:
9         print ('Se recorrió el diccionario y NO se agregaron valores')
10    print ('El diccionario al final del proceso es: ', dic)
11
```

¿Se ejecuta o no la cláusula **else**? Lo vemos

---

## Resultados de ejecutar y no ejecutar la cláusula else



```
C:\Windows\system32\cmd.exe
Juan
Ana
El diccionario al final del proceso es: <1: 'Juan', 2: 'Ana', 5: 'Helena', 3: 'Agregado'>
Presione una tecla para continuar . . .
```

No se ejecuta la cláusula **else**

Y si termina normal?

Probamos?

---

## Al terminar en forma normal el bloque...

Se ejecuta la cláusula **else**



```
C:\Windows\system32\cmd.exe
Juan
Ana
Helena
Se recorrio el diccionario y NO se agregaron valores
El diccionario al final del proceso es: {1: 'Juan', 2: 'Ana', 5: 'Helena'}
Presione una tecla para continuar . . .
```

"else" cláusula opcional que se ejecuta **SOLO** si **NO** se levanta una excepción en el bloque try except

## Cláusulas finally

```
1 def dividir (x,y):
2     try:
3         return x / y
4
5     except ZeroDivisionError:
6         print ('Division por CERO!')
7         return 'Indefinido'
8     finally:
9         print('Ha FINALIZADO la ejecucion del modulo')
10
11 x = int(input('Ingresa el dividendo: '))
12 y = int(input('Ingresa el divisor: '))
13 result = dividir(x,y)
14 print ('La division es', result)
15
```

¿Se ejecuta o no la cláusula **finally**? Lo ejecutamos?

## Al no terminar en forma normal el bloque...

```
C:\WINDOWS\SYSTEM32\cmd.exe
Ingresa el dividendo: 8
Ingresa el divisor: 0
Division por CERO!
Ha FINALIZADO la ejecucion del modulo
La division es Indefinido

-----
(program exited with code: 0)
Presione una tecla para continuar . . .
```

Se ejecuta la cláusula **finally**

Lo ejecutamos de nuevo?

## Al terminar en forma normal el bloque...

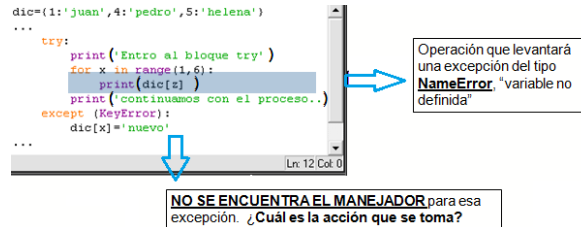
También se ejecuta la cláusula **finally**

```
C:\WINDOWS\SYSTEM32\cmd.exe
Ingresa el dividendo: 8
Ingresa el divisor: 2
Ha FINALIZADO la ejecucion del modulo
La division es 4.0

-----
(program exited with code: 0)
Presione una tecla para continuar . . .
```

"**finally**" cláusula que se ejecuta **SIEMPRE**, independientemente de que se haya levantado o no una excepción en el bloque try except

## ¿Qué sucede en estos casos?



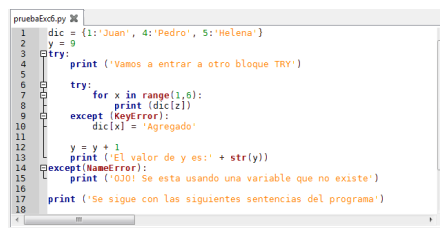
class: destacado

## Debemos indagar sobre ...

La forma de **propagación** que utiliza Python.

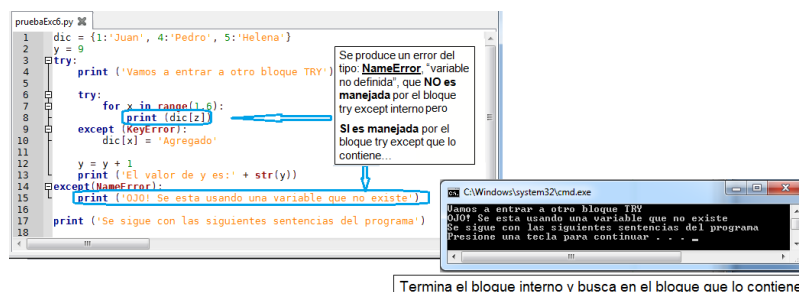
- Cuando no encuentra un manejador para la excepción levantada, ¿dónde busca?

## Busca primero estáticamente



Lo comprobamos?

## ¿Qué hizo?



- ¿Qué pasó con las sentencias..?

```
y = y + 1
print ('El valor de y es:' + str(y))
```

---

**class:** destacado

## El ejemplo demuestra que...

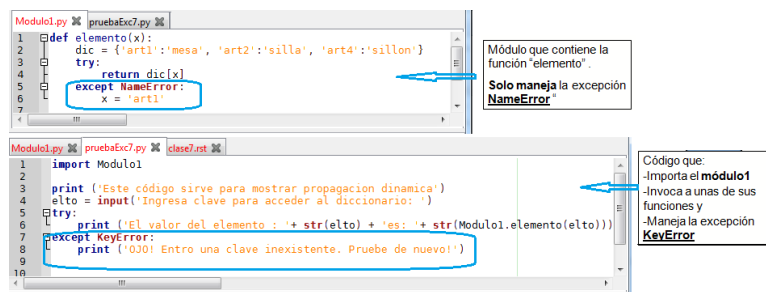
Python aplica el mecanismo de **terminación**

- Y.. ¿qué sucede si no encuentra un manejador?

---

## Busca dinámicamente

- Supongamos..

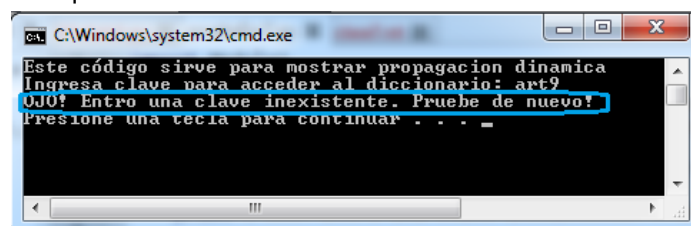


¿Qué sucede?

---

## Sucedio que...

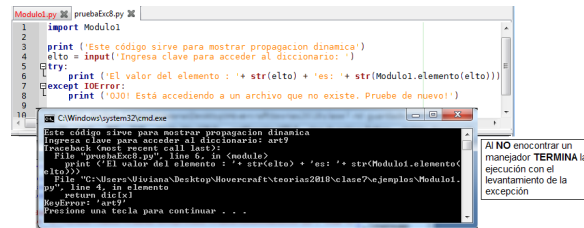
- El error se levantó dentro de la función **elemento** del **Modulo1.py**
- Al no manejarse allí la excepción y, al no encontrar un manejador para esa excepción..
- Bajó dinámicamente a quién lo llamó.



---

## ¿Y sino encuentra un manejador?...

Se aborta el programa

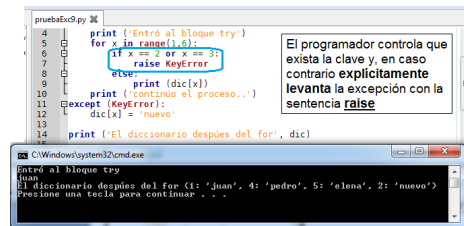


```
1 import Modulo1
2
3 print ('Este código sirve para mostrar propagación dinamica')
4 elto = input('Ingresa clave para acceder al diccionario: ')
5
6 try:
7     print ('El valor del elemento : ' + str(elto) + ' es: ' + str(Modulo1.elemento(elto)))
8 except IOError:
9     print ('OJO! Está accediendo a un archivo que no existe. Pruebe de nuevo!')
```

Al NO encontrar un manejador TERMINA la ejecución con el levantamiento de la excepción

## Levantando excepciones "explícitamente"

- El programador puede levantar explícitamente una excepción con la sentencia **raise**



```
4 print ('Entró al bloque try')
5 for x in range(1,6):
6     if x == 2 or x == 3:
7         raise KeyError
8     else:
9         print (dic[x])
10    print ('continúo el proceso...')
11 except (KeyError):
12    dic[x] = 'nuevo'
13
14 print ('El diccionario después del for', dic)
```

El programador controla que exista la clave y, en caso contrario explícitamente levanta la excepción con la sentencia **raise**

**class:** destacado

## Excepciones predefinidas (Built-in)

- Algunas de ellas:
- **EOFError**: errores de EOF.
- **ImportError**: error con importación de módulos.
- **ModuleNotFoundError**: error por módulo no encontrado.
- **IndexError**: error por índice fuera de rango.
- **KeyError**: error por clave inexistente.
- **NameError**: error por nombre no encontrado.
- **SyntaxError**: error por problemas sintácticos
- **ZeroDivisionError**: error por división por cero.
- **IOError**: error en entrada salida.
- Etc.

El programador puede definir nuevas excepciones definiendo clases que deriven de **Excpetion**.

## ¿Retomamos el juego del ahorcado?

- Pensemos en modificarlo para hacerlo más personalizado.
- El programa debería poder levantar los temas y palabras desde un archivo.

- ¿Retomamos el último modificado (ahorcado con funciones)?

## Y modificamos...

Esta parte del código ...

```
#Defino conjunto de palabras a trabajar por temas
palabras = {1:['gato', 'perro','pato','elefante','lobo'],
            2:['rojo','azul','verde','amarillo'],
            3:['milanesa','pure','pizza','salchicha']}

def defino_palabra(palabras):
    ''' Esta función retorna la palabra a adivinar.'''

    #Pido que el jugador elija un tema
    tema = int(input('Elige un tema:\n 1: animales\n 2: colores\n 3: comidas\n '))

    #Selecciono la palabra a trabajar
    pal = palabras[tema][random.randrange(len(palabras[tema]))]

    return pal
```

## Haciendo lo siguiente...

Colocamos al comienzo la sentencia **import json** y..

```
# Eliminamos la definición del diccionario "palabras"
# La función defino_palabra NO recibe parámetros (palabras) porque se crea
# el diccionario adentro de la función
def defino_palabra():
    ''' Esta función retorna la palabra a adivinar.'''

    # Abro archivo de palabras
    conj_palabras= open('palabrasJuego.txt','r')
    palabras = json.load(conj_palabras)

    # Pido que el jugador elija un tema. CAMBIADO porque ahora el tema depende
    # de lo que se levantó del archivo

    print ('Elige un tema:')
    for tema in palabras:
        print('\t', tema, ': ', palabras[tema][0])

    #CAMBIO porque ahora la clave del archivo es texto.
    tema=input('Ingresa la opción: ')

    #Selecciono la palabra a trabajar
    pal = palabras[tema][1][random.randrange(len(palabras[tema][1]))]

    return pal
```

Quitamos la inicialización de la variable "palabras"

Quitamos el parámetro de la función dado que ahora NO recibe el diccionario "palabras"

Leemos los temas y palabras

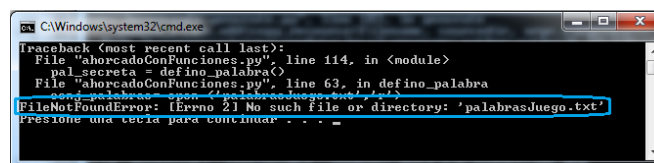
Mostramos opción y temática

Elegimos la palabra

Vemos si funciona?

## ¿Y qué pasa si no encuentra el archivo?...

Se levanta una **excepción**



## ¿Solución?

Con **manejo de excepciones**

```
def defino_palabra():
    ''' Esta función retorna la palabra a adivinar. '''
    #Encierro la lectura del archivo en un bloque try except
    try:
        # Abro archivo de palabras
        con_palabras = open('palabrasJuego.txt', 'r')
        palabras = json.load(con_palabras)
        # En caso de levantarse la excepción inicializo palabras con datos fijos
        # Observar que se cambiaron algunas cosas:
        # 1- La clave es cadena e incorporamos
        # 2- Incorporamos el tema
    except:
        palabras = {
            '1': ['animales', ['gato', 'perro', 'pato', 'elefante', 'lobo']],
            '2': ['colores', ['rojo', 'azul', 'verde', 'amarillo']],
            '3': ['comidas', ['milanesa', 'pure', 'pizza', 'salchicha']]
        }

    # Pido que el jugador elija un tema. CAMBIADO porque ahora el tema depende
    # de lo que se levanto del archivo

    print('Elije un tema:')
    for tema in palabras:
        print('\t', tema, ': ', palabras[tema][0])

    #CAMBIO porque ahora la clave del archivo es texto.
    tema = input('Ingresa la opción: ')

    #Selecciono la palabra a trabajar
    pal = palabras[tema][1][random.randrange(len(palabras[tema][1]))]

    return pal
```

Se encierra las sentencias de lectura entre try y except

En caso de levantamiento de la excepción se inicializa la variable "palabras" con valores por defecto.

¿Vemos el [ejemplo](#)?

**class:** destacado

## ¿Y que pasa si ingreso una opción no listada?

Pensar en la solución..

**class:** destacado

## ¿Y si lo implementamos con PySimpleGUI?

¿Lo vemos?..

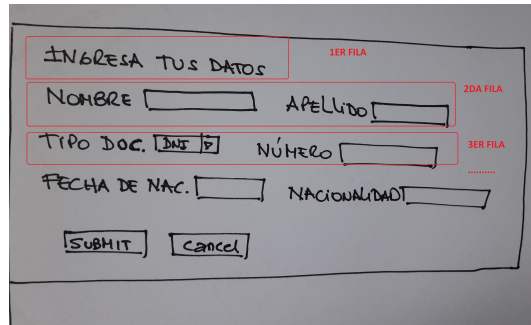
## PySimpleGUI: Algunas cosas más ...

¿Qué pudimos ver en el programa?

- Diseño de la interface presentado en columnas
- Procesamiento de los datos ingresados

## Capa Layout: diseño de la interface

- Para diseñar la interface que colocaremos sobre una ventana debemos:
  - Armar una **lista de listas**.
  - Cada elemento de la lista contendrá un lista con todos los elementos que van a estar en una línea
  - Cada elemento es a su vez una lista



---

## Armando un diseño para esa interface

```
import PySimpleGUI as sg
#Armo el diseño de la interface
diseño = [
    [sg.Text('Ingresa tus datos')],
    [sg.Text('Nombre'), sg.InputText(), sg.Text('Apellido'), sg.InputText() ],
    [sg.Text('Tipo Doc'), sg.Listbox(values=('DNI', 'LE', 'PAS'), size=(30, 1)), sg.Text('Número'), sg.InputText()],
    [sg.Text('Fecha de Nac'), sg.InputText(), sg.Text('Nacionalidad'), sg.InputText()],
    [ sg.Submit(), sg.Cancel()]
]

#La aplico a la ventana y la muestro
window = sg.Window('Datos Personales').Layout(diseño)
window.Read()
```

¿Vemos como queda?

---

## Armando un diseño para esa interface

¿Así como está ... puedo manipular los datos ingresados por el usuario?

---

## Trabajando con la información ingresada por el usuario

- `window.Read()` no solo muestra la ventana sino que retorna información de las acciones por el usuario
    - Devuelve dos valores:
      - El **texto** del botón que se cliqueo
      - Una **lista** de valores que el usuario ingresó
- 

## Trabajando con la información ingresada por el usuario

- Deberíamos entonces:

- Almacenar esos datos para poder manipularlos.
- En el código anterior, la última sentencia la podríamos cambiar por:

```
(boton_clickeado, datos_ingresados) = window.Read()
```

Lo probamos y vemos qué información nos **da**?

---

## Mejorando el diseño de interface

- Supongamos que ahora queremos mostrar los datos de la siguiente forma:

- Donde los elementos no están dispuestos en forma de líneas sino en forma de columnas
  - Se deberá armar el diseño para cada columna
  - Con el uso del elemento **Column** se incorporan al diseño general de la ventana
- 

**class:** destacado

## Elemento Columna

`Column(Diseño_Columna, background_color= color)`

---



## Armando del diseño con columnas

```
import PySimpleGUI as sg
#Armando una columna

columna_1 = [ [sg.Text('Nombre'), sg.InputText()],
               [sg.Text('Apellido'), sg.InputText()] ]

columna_2 = [ [sg.Text('Tipo Doc'), sg.Listbox(values=('DNI', 'LE', 'PAS'), size=(30, 1)),
               [sg.Text('Número'), sg.InputText()],
               [sg.Text('Nacionalidad'), sg.InputText()] ]

#Armo el diseño de la interface
diseño = [
    [sg.Text('Ingresa tus datos')],
    [sg.Column(columna_1), sg.Column(columna_2)],
    [ sg.Submit(), sg.Cancel()]
]
#La aplico a la ventana y la muestro
window = sg.Window('Datos Personales').Layout(diseño)
window.Read()
```

¿Cómo se verá [ahora](#)?

---

## ¿Se animan?

- ¿Se animan a hacer el ahorcado usando PySimpleGUI?
- 

## Resumen

- Repaso entrada salida de archivos
- Listas por comprensión
- Manejo de excepciones
- Más sobre PySimpleGUI
- Avanzamos en PySimpleGUI