title: Presentacion Python 2019
Author: Claudia Banchoff, Viviana Harari
description: clase 4
keywords: definición de funciones y parámetros
css: estilo.css

Seminario de Lenguajes - Python

Cursada 2019

Temario

- Repaso de la clase anterior.
- Un poco más sobre funciones.
- Expresiones lambda.
- GUI.

Repaso: ¿qué vimos hasta ahora?

- Sintaxis del lenguaje
    - Uso de variables
    - Tipos de datos
    - Estructuras de control
- ¿Cómo modularizamos nuestros programas?

Repaso: funciones en Python

```
def nombre_funcion(parametros):
        sentencias
        return <expresion>
```

- ¿Cómo pasamos parámetros?
- ¿Cómo retorno valores? ¿Uno solo?

Veamos el siguiente ejemplo

Queremos definir una función que, dada una cadena de caracteres, retorne la **cantidad de vocales abiertas, vocales cerradas y la cantidad total de caracteres** de la misma.

```
def retorno_varios(cadena):
        cant_aeo = cadena.count("a") + cadena.count("e") + cadena.count("o")
        cant_iu = cadena.count("i") + cadena.count("u")
        return (cant_aeo, cant_iu, len(cadena))


algo = retorno_varios("Seminario de Python")
```

- ¿Qué tipo de dato retorna la función?

```
print(type(algo))
```

En este mismo ejemplo

Queremos definir una función que, dada una cadena de caracteres, retorne la **cantidad de vocales abiertas, vocales cerradas y la cantidad total de caracteres** de la misma.

```
def retorno_varios(cadena):
        cant_aeo = cadena.count("a") + cadena.count("e") + cadena.count("o")
        cant_iu = cadena.count("i") + cadena.count("u")
        return (cant_aeo, cant_iu, len(cadena))
```

- **¿Es correcto el siguiente código?**

```
vocales_abiertas = retorno_varios("Seminario de Python")[0]
```

Parámetros en Python

- Se acuerdan de la función **esperar** del juego del bosque.py:

```
def esperar(tiempo):
        time.sleep(tiempo)
```

- Vamos a reformularla:

```
def esperar(tiempo=2):
        time.sleep(tiempo)
```

```
esperar()
esperar(1)
```

- ¿Qué notan?
- En Python, los parámetros pueden tener **valores por defecto**.

Y acá, ¿qué pasa?

```
import time

def imprimo_mensaje(mensaje, repito=2, tiempo=1):
        for i in range(0,repito):
                print(mensaje)
                time.sleep(tiempo)

imprimo_mensaje("Quiero irme ya!!!")
imprimo_mensaje("Quiero irme ya!!!", 2, 4)
imprimo_mensaje("Quiero irme ya!!!", 4)
imprimo_mensaje("Quiero irme ya!!!", tiempo=2, repito=4)
```

- Puedo invocar a la función con los parámetros en **otro orden** pero **nombrando al parámetro**.
- Si no lo nombro, asume el orden posicional.
- Si hay más de un argumento, los que tienen **valores por defecto siempre van al final de la lista de parámetros**.

¡¡ATENCIÓN!!

- Miremos este código y veamos qué imprime:

```
def funcion(a, L=[]):
        L.append(a)
        return L

print(funcion(1))
print(funcion(2))
print(funcion(3))
```

---

class: destacado

¡¡ATENCIÓN!!

Importante: los valores por defecto se evalúan **UNA única vez** en la definición de la función.

Problema

Quiero escribir una función que me permita imprimir un número variable de parámetros

¿Cómo se les ocurre resolverlo?

Número variable de argumentos

- Python permite definir funciones con un **número variable de parámetros**.

```
def sumo_parametros(*args):
        return sum(args)

sumo_6 = sumo_parametros(1,2,3,4,5,6)
sumo_3 = sumo_parametros(23,1,2)
```

- **args** es una **tupla** que representa a los parámetros pasados.

Seguimos con parámetros

- Otra forma de definir una función con un número variable de parámetros.

```
def imprimo_valores(**kwargs):
    for clave, valor in kwargs.items():
        print("El valor de {} es {}".format(clave, valor))

imprimo_valores(
        nom1 = 'Batmam',
        nom2 = 'Capitana Marvel',
        nom3 = 'Ironman'
        )
```

- **kargs** es una **diccionario** que representa a los parámetros pasados.

¿Y acá?

¿Qué estoy pasando como parámetro?

```
def imprimo_agenda(nombre, celu):
        print(nombre, celu)


contactos = {"nombre": "Juan", "celu": 12345}
imprimo_agenda(**contactos)
```

- Veamos estos otros ejemplos de parámetros

¿Todo junto se puede?

- Ejemplo sacado de https://docs.python.org/3/tutorial/controlflow.html#defining-functions

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])
```

- Lo invocamos...

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper="Michael Palin",
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

Variables locales y globales

```
x = 12
a = 13
def funcion(a):
        x = 9
        a = 10
```

- Variables locales enmascaran las globales.
- Acceso a las globales mediante **global**.

Variables locales y globales

```
x = 12
a = 13
def funcion(a):
        global x
        x = 9
        a = 10
```

- No es una buena práctica de programación utilizar variables globales.

Variables locales y globales

- Analicemos este ejemplo

```
x = 12
def funcion1():
        temp = x + 1
        print(temp)

def funcion2():
        x = x + 1
        print(x)
```

- ¿Qué les parece que pasa si invocamos a las dos funciones?

Volvamos a los parámetros

- ¿Qué les parece que imprime este código?¿Por qué?

```
i = 4
def funcion(x=i):
        print(x)

i = 10
funcion()
```

Funciones anidadas

- Python permite definir una función dentro de otra.

- Debe estar definida antes de usarse.
    - Por legibilidad, nosotros la vamos a definir siempre al principio de cada función o módulo.
- Veamos este ejemplo

Un poco más sobre variables globales

```
x = 222222
def uno():
        x = 10
        def dos():
                nonlocal x
                print(x)
                x += 1
        dos()
        print(x)
uno()
```

- ¿nonlocal?
- Nuevamente: no es una buena práctica de programación utilizar variables globales, pero necesitamos conocer las reglas de alcance.

Algo más sobre funciones

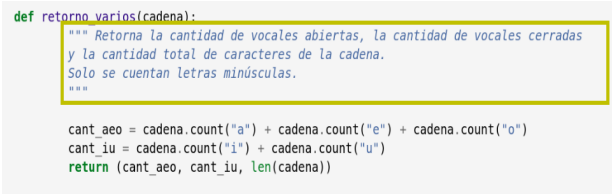
- Reescribimos la función **retorno_varios**.

```
def retorno_varios(cadena):
        """ Retorna la cantidad de vocales abiertas, la cantidad de vocales cerradas
        y la cantidad total de caracteres de la cadena.
        Solo se cuentan letras minúsculas.
        """

        cant_aeo = cadena.count("a") + cadena.count("e") + cadena.count("o")
        cant_iu = cadena.count("i") + cadena.count("u")
        return (cant_aeo, cant_iu, len(cadena))
```

- ¿Notan algo extraño?

Las funciones tienen atributos




```
print(retorno_varios.__doc__)
print(retorno_varios.__defaults__)
print(retorno_varios.__name__)
```

- **funcion.__doc__**: es una cadena que contiene un texto que aparece en la primer línea de la definición de la función: **Docstring**.
- **funcion.__name__**: es una cadena con el nombre la función.
- **funcion.__defaults__**: es una tupla con los valores por defecto de los parámetros opcionales.

¿Ordenamos una lista?

- Ya vimos el método **sort** y la función **sorted()**.

```
def uso_sort(cadena):

        lista = cadena.split()

        print("Teniendo en cuenta mayúsculas y minúsculas")
        lista.sort()
        print(lista)

        lista.sort(key=str.lower)
        print("Sin tener en cuenta mayúsculas y minúsculas")
        print(lista)

def uso_sorted(cadena):

        print(sorted(cadena.split(), key=str.lower))

uso_sort("Hoy puede ser un gran día. ")
uso_sorted("Hoy puede ser un gran dia")
```

Seguimos ordenando

- Miremos este otro ejemplo de **sorted**:

```
def uso_sorted1():
        usuarios_juego = [
                ('Juan', 'Nivel1', 15),
                ('Maria', 'Nivel1', 12),
                ('Jose', 'Nivel2', 1020),
                ('Ana', 'Nivel2', 1020),
        ]
        return sorted(usuarios_juego, key=lambda usuario: usuario[0])

print(uso_sorted1())
```

- ¿Qué les parece que estamos intentando hacer?
- ¿lambda?

Expresiones lambda

- ¿Qué son las expresiones lambda?
    - Funciones sencillas.
    - Anónimas.
- Forma general: **lambda** parametros **:** expresion

```
lambda a, b: a*b
lambda a, b=1: a*b
```

Expresiones lambda

```
lambda a, b=1: a*b
```

- Es equivalente a:

```
def producto(a, b=1):
        return a*b
```

- Una expresión lambda es una función **anónima**, por lo tanto el nombre de la función (**producto**) no está presente.

Expresiones lambda

- Algunos ejemplos de uso

```
lista = [lambda x: x * 2, lambda x: x * 3]
param = 4
for accion in lista:
        print(accion(param))
```

- ¿Qué tipo de elementos contiene la lista?
- ¿Qué imprime?

Expresiones lambda

- Un ejemplo de la documentación oficial.

```
def make_incrementor (n):
        return lambda x: x + n

f = make_incrementor(2)
g = make_incrementor(6)

print (f(42), g(42))
print (make_incrementor(22)(33))
```

Veamos algunos ejemplos de uso

- Funciones **map** y **filter**

```
def doble(x):
        return 2*x

def es_par(x):
        return x%2 == 0

lista  = [1, 2, 3, 4, 5, 6, 7]

print(list(map(doble, lista)))
print()
print(list(filter(es_par, lista)))
```

Ahora usamos expresiones lambda

```
lista  = [1, 2, 3, 4, 5, 6, 7]

print(list(map(lambda x: 2*x, lista)))
print()
print(list(filter(lambda x: x%2 == 0, lista)))
```

¿Tenían una tarea?

El ahorcado con funciones

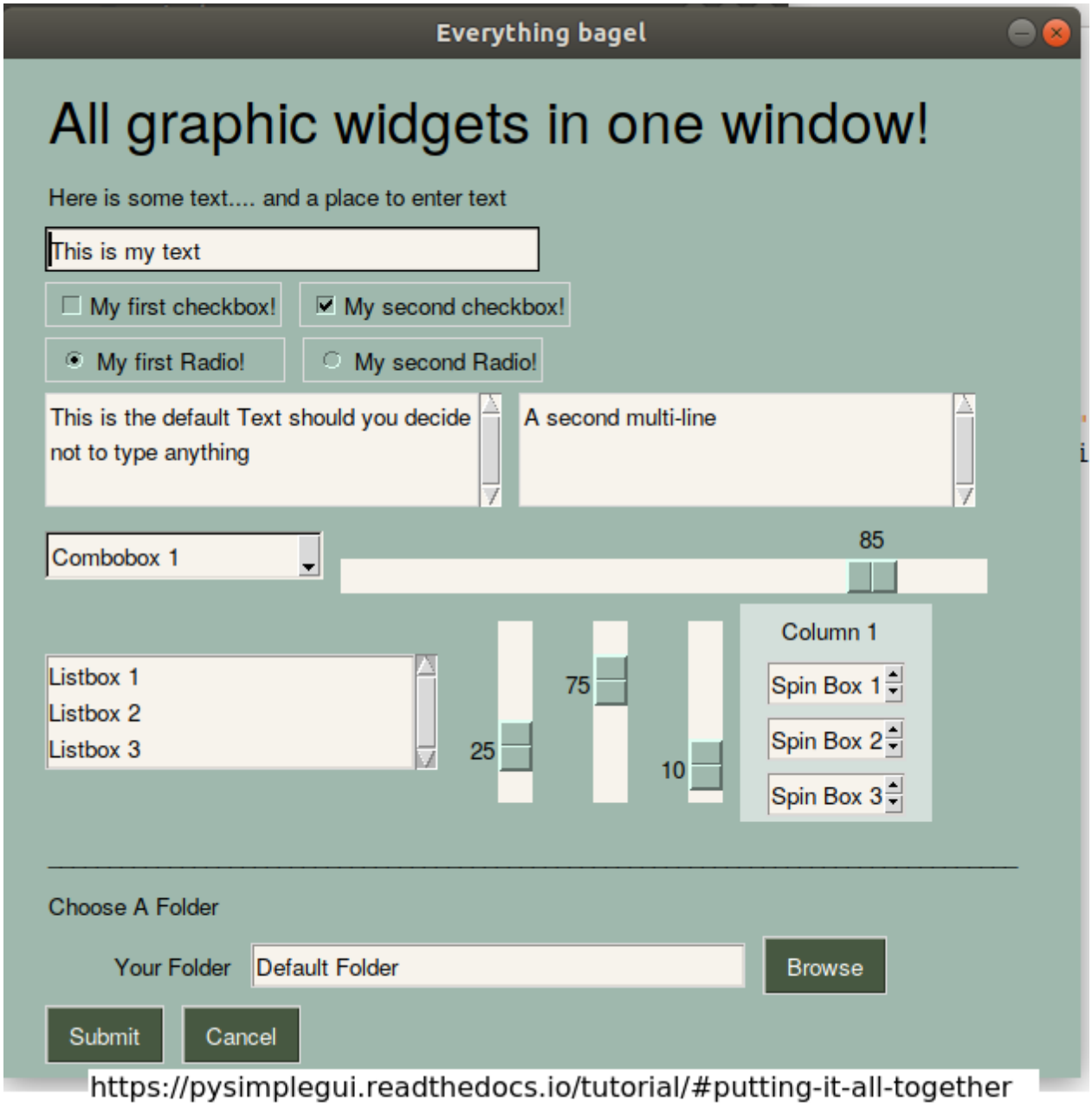
- Una solución podría ser: ahorcadoConFunciones.py

¿La mejoramos?

- Agregar niveles de dificultad.
- ¿Mayúsculas y minúsculas?
- ¿Una palabra al azar sin especificar el tema?
- Pensemos:
    - ¿Dónde podríamos sacar provecho de las características de las funciones de Python?
        - Valores por defecto en los parámetros.
        - Uso de expresiones lambda.
- **Por Python plus**, me pueden enviar su versión del juego del ahorcado con funciones.
- **Tarea habilitada en catedras.info hasta el miércoles 10 las 23:59 hs.**

¿Saben qué son las GUI?

GUI: Graphical User Interface

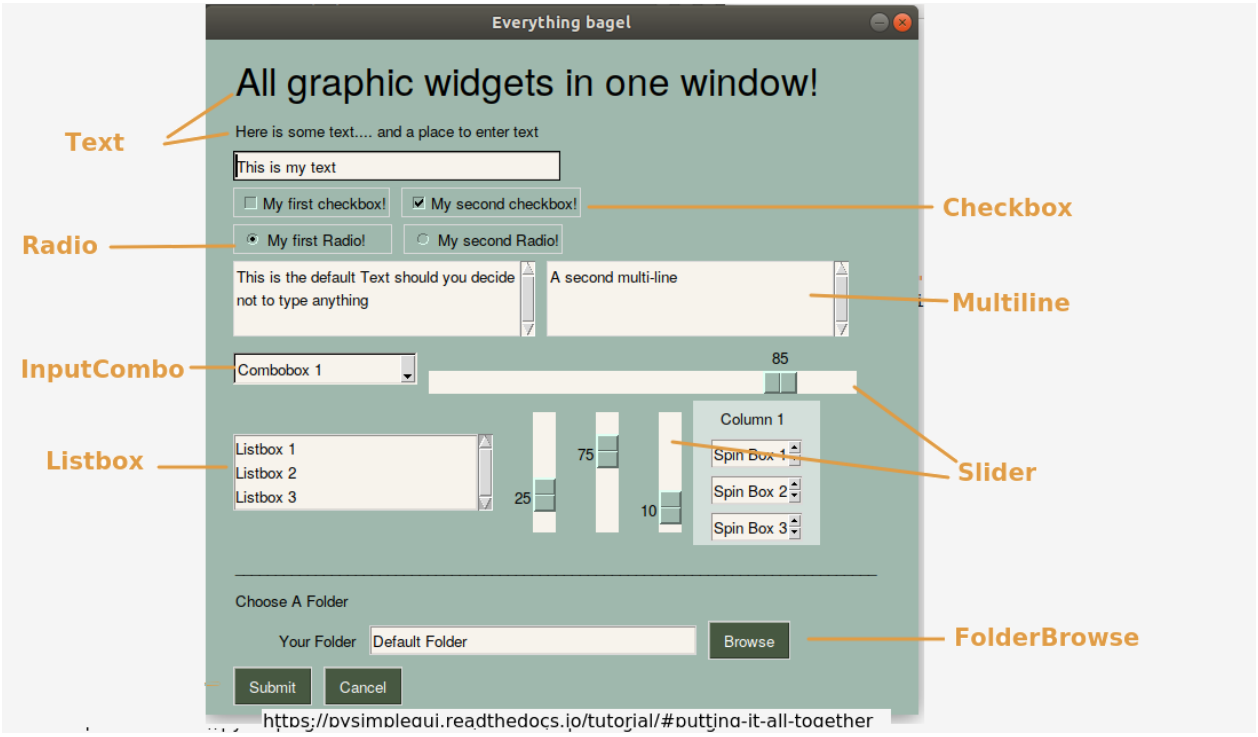


GUI en Python

- Vamos a usar PySimpleGUI
- Es un **framework** bastante simple para desarrollar interfaces gráficas en Python.
- Es software libre: https://github.com/PySimpleGUI/PySimpleGUI
- Se instala con pip.
- Vamos a ir de a poco...

- Más info en: https://pysimplegui.readthedocs.io/tutorial

¿Con qué elementos podemos trabajar?



Popups: las ventanas más sencillas

- Observemos el código

```
import PySimpleGUI as sg
sg.Popup('Mi primera ventanita')
```

- **import PySimpleGUI as sg**, permite acceder a los recursos por el nombre **sg**.
- El ejemplo completo de ventanas popups.

¿Cómo organizamos la UI?

- Observemos el código:

```
import PySimpleGUI as sg

layout = [[sg.Text('¡¡ Hola Mundo!!!')],
          [sg.OK()] ]

window = sg.Window('Pequeña ventanita').Layout(layout)

evento = window.Read()
window.Close()
```

- ¿layout?, ¿window?

Layout

- Representa al esquema o diseño de nuestra UI.
    - Cómo se distribuyen los elementos en la interfaz.

```
layout = [[sg.Text('¡¡ Hola Mundo!!!')],
          [sg.OK()] ]
```

- ¿De qué tipo es?
- ¿Qué elementos incluimos en este ejemplo?

Elementos de la UI

- Acá van algunos disponibles en PySimpleGUI
    - Buttons: File Browse, Folder Browse, Color chooser, Date picker, etc.
    - Checkbox, Radio Button, Listbox
    - Slider, Progress Bar
    - Multi-line Text Input, Scroll-able Output
    - Image, Menu, Frame, Column, Graph, Table

Agreguemos elementos

Miremos este ejemplo que tiene los elementos básicos.

---

class: destacado

PySimpleGUI

- En PySimpleGUI.py están definidas todos elementos y constantes que podemos utilizar.
- Por ejemplo, queremos agregar el elemento "Color chooser".
- Encontramos el componente **ColorChooserButton**:

```
def ColorChooserButton(button_text, target=(None, None), image_filename=None, image_data
                       image_size=(None, None), image_subsample=None, tooltip=None, bord
                       size=(None, None), auto_size_button=None, button_color=None, disa
                       font=None, bind_return_key=False, focus=False, pad=None, key=None
```

- ¿Cuántos parámetros obligatorios requiere?
- Observemos los valores por defecto de los restantes parámetros.

¡¡Las ventajas del software libre!!

Ahora a aplicar todo en la práctica

¿Qué vimos hoy?

Resumen

- Más sobre funciones.
- Expresiones lambda.
- GUI.
- Las ventajas del software libre.