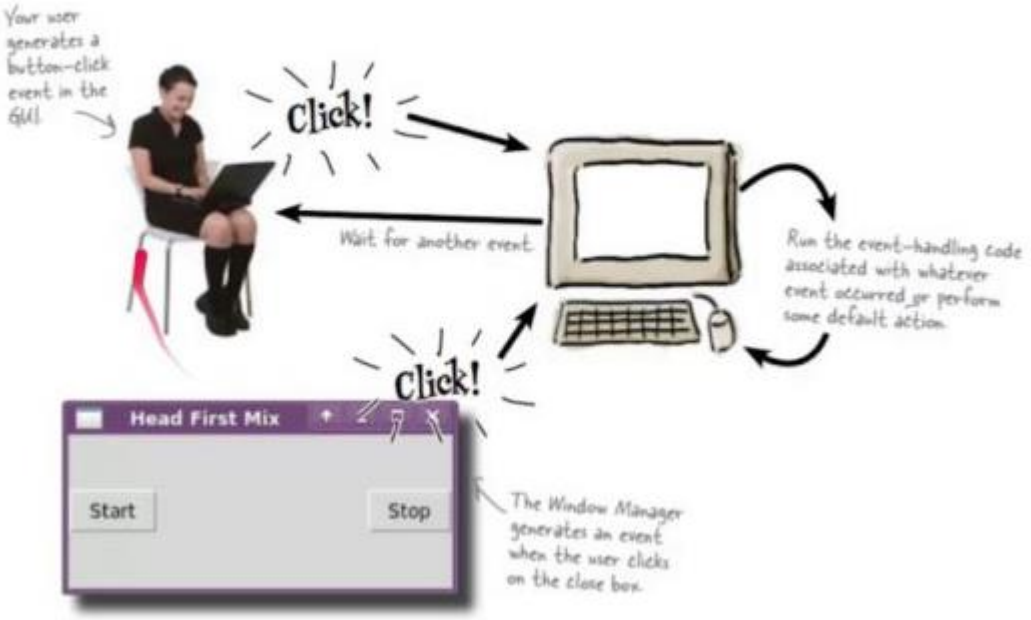
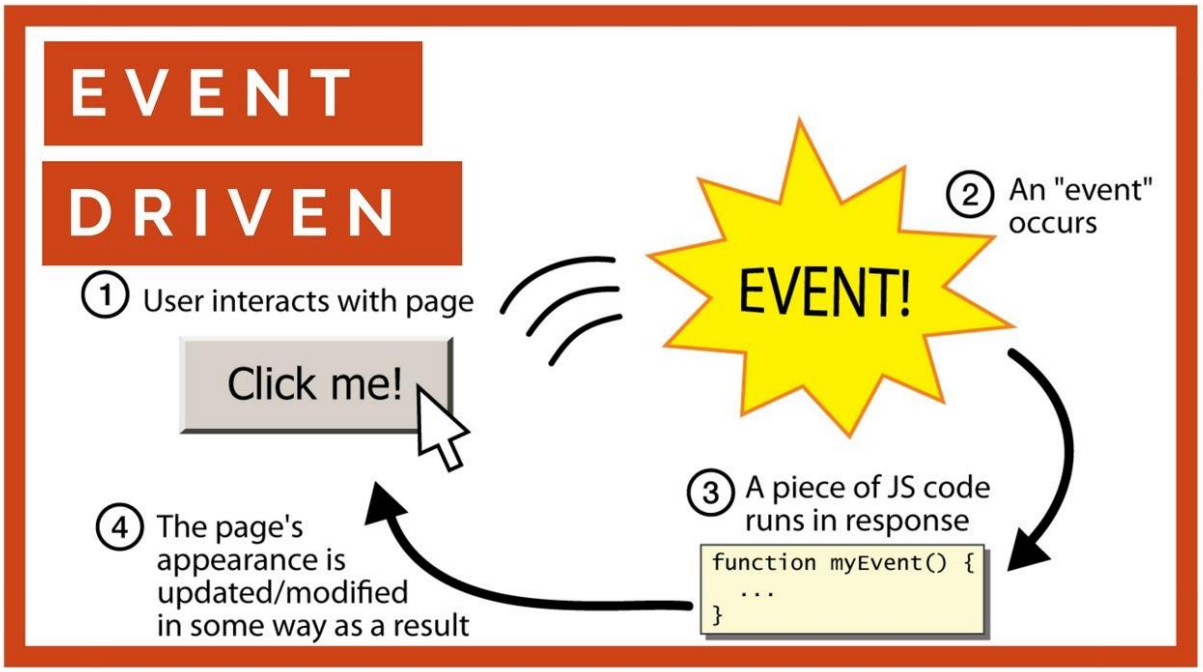
Unió Europea
Fons social europeu

Programación Orientada a Eventos



La programación dirigida por eventos es un paradigma de programación en el que tanto **la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.**

Mientras en la programación secuencial (o estructurada) es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos será el propio usuario —o lo que sea que esté accionando el programa— el que dirija el flujo del programa.

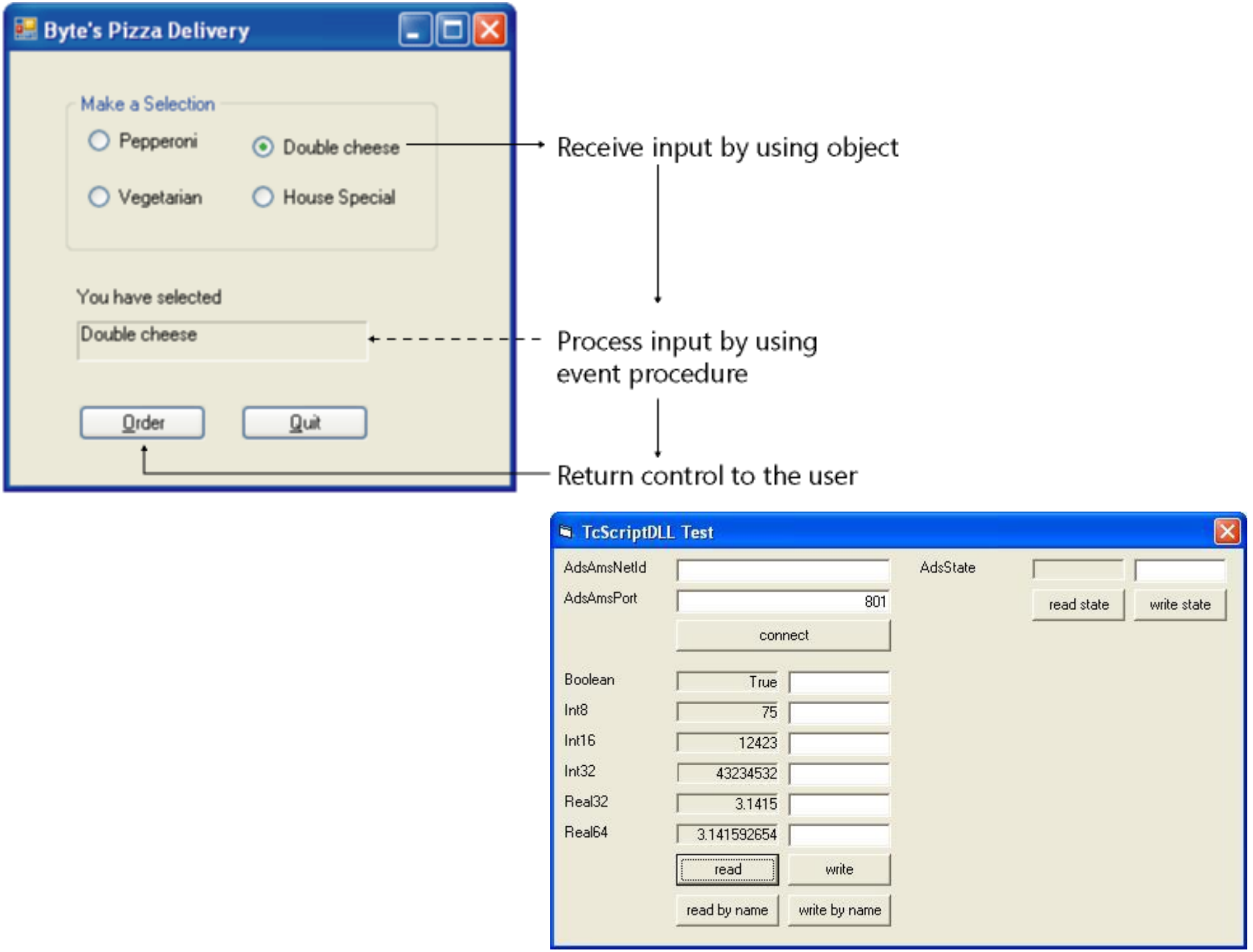


1. Administrador de eventos

El creador de un programa dirigido por eventos debe definir los eventos que manejarán su programa y las acciones que se realizarán al producirse cada uno de ellos, lo que se conoce como el administrador de evento.

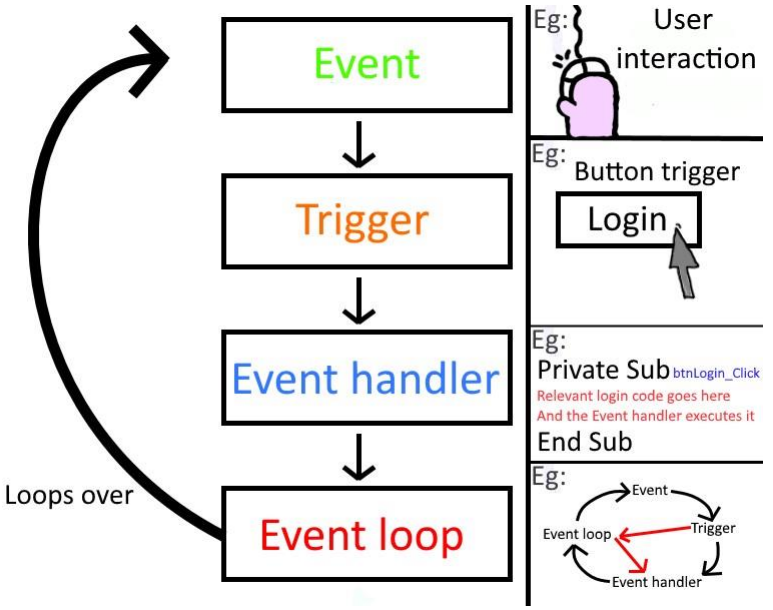
Los eventos soportados estarán determinados por el lenguaje de programación utilizado, por el sistema operativo e incluso por eventos creados por el mismo programador.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará bloqueado hasta que se produzca algún evento


Byte's Pizza Delivery
Make a Selection
Pepperoni
Double cheese
Vegetarian
House Special
You have selected
Double cheese
Order
Quit
Receive input by using object
Process input by using event procedure
Return control to the user
TcScriptDLL Test
AdsAmsNetId
AdsAmsPort
801
connect
AdsState
read state
write state
Boolean
True
Int8
75
Int16
12423
Int32
43234532
Real32
3.1415
Real64
3.141592654
read
write
read by name
write by name

Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código que esté asociado al evento.

Por ejemplo, si el evento consiste en que el usuario ha hecho clic en el botón de **play** de un reproductor de películas, se ejecutará el código del administrador de evento, que será el que haga que la película se muestre por pantalla.



La programación dirigida por eventos es la base de lo que llamamos interfaz de usuario, aunque puede emplearse también para desarrollar interfaces entre componentes de Software o módulos del núcleo.

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

2. Detección de eventos

En contraposición al modelo clásico, la programación orientada a eventos permite interactuar con el usuario en cualquier momento de la ejecución.

Esto se consigue debido a que los programas creados bajo esta arquitectura se componen por un **bucle exterior permanente** encargado de recoger los eventos, y **distintos procesos** que se encargan de tratarlos.

Habitualmente, este bucle externo permanece oculto al programador que simplemente se encarga de tratar los eventos, aunque en algunos entornos de desarrollo (IDE) será necesaria su construcción.

2.1. Problemática

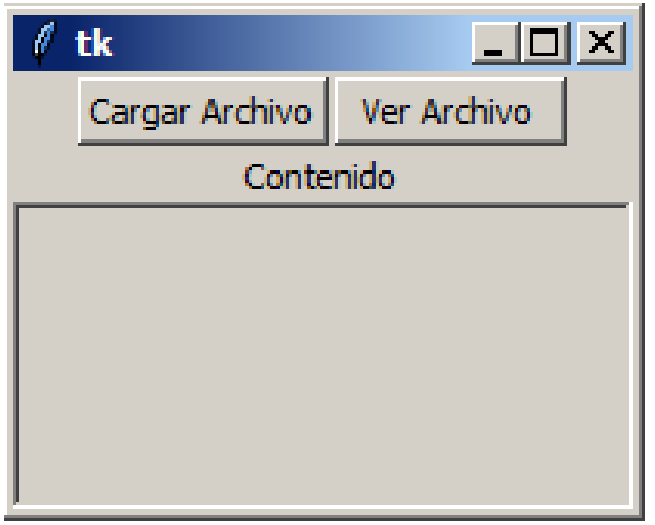
El flujo (orden) de ejecución del software escapa al control del programador.

En cierta manera podríamos decir que en la programación clásica el flujo estaba en poder del programador y era este quien decidía el orden de ejecución de los procesos, mientras que en programación orientada a eventos, es el usuario el que controla el flujo y decide.

Ejemplo: un menú con dos botones : Cargar y Ver.

- Cuando el usuario pulsa el botón “Cargar”, el programa se encarga de recoger ciertos parámetros que están almacenados en un fichero y calcular algunas variables.

- Cuando el usuario pulsa el botón “Ver”, se muestran por pantalla dichas variables



Es sencillo darse cuenta de que la naturaleza indeterminada de las acciones del usuario y las características de este paradigma pueden fácilmente desembocar en el error fatal de que se pulse el botón “Ver” sin previamente haber sido pulsado el botón “Cargar”.

Aunque esto no pasa si se tienen en cuenta las propiedades de dichos botones, haciendo inaccesible la pulsación sobre el botón “Ver” hasta que previamente se haya pulsado el botón “Cargar”.

```
button.state(['disabled']) # set the disabled flag, disabling the button
button.state(['!disabled']) # clear the disabled flag
button.instate(['disabled']) # return true if the button is disabled,else false
button.instate(['!disabled']) # return true if the button is not disabled, else false
button.instate(['!disabled'], cmd) # execute 'cmd' if the button is notdisabled
```

3 GUI’s / Interfaces Gráficas de Usuarios

Las GUI (Graphical User Interface), han acercado la informática al **usuario no-experto**.

Utilizar el software de manera intuitiva y sin necesidad de grandes conocimientos, ha colaborado a mejorar la productividad en muchas tareas.

El **ratón** cuenta con sus propios eventos.

Actualmente los móviles y otros dispositivos han modificado las listas de eventos posibles …

Vinculando Eventos en Python

Una aplicación Tkinter pasa la mayor parte de su tiempo dentro de un bucle de eventos (donde entra a través del método **mainloop**).

Los eventos pueden provenir de varias fuentes, incluidas las pulsaciones de teclas y las operaciones del mouse por parte del usuario, y volver a dibujar los eventos desde el administrador de ventanas (en muchos casos, causado indirectamente por el usuario).

Tkinter proporciona un mecanismo poderoso para permitirle lidiar con los eventos usted mismo.
Para cada widget, puede **enlazar** funciones y métodos de Python a eventos.

widget.bind (evento, manejador)

Si se produce un evento que coincide con la descripción del *evento* en el widget, se llama al *controlador* dado con un objeto que describe el evento.

Ejercicio : Prueba este programa que detecta eventos del ratón.

```
from tkinter import *
root = Tk()
root.geometry('500x300')
l =Label(root, text="Zona cero...", bg="green", fg="white",
font=("Verdana",18))
l.grid()
l.bind('<Enter>', lambda e: l.configure(text='Moved mouse inside'))
l.bind('<Leave>', lambda e: l.configure(text='Moved mouse outside'))
l.bind('<1>', lambda e: l.configure(text='Clicked left mouse button'))
l.bind('<Double-1>', lambda e: l.configure(text='Double clicked'))
l.bind('<B3-Motion>', lambda e: l.configure(text='right button drag to
%d,%d' % (e.x, e.y)))
root.mainloop()
```

Ejercicio : Algunas de las funciones lambda han sido substituidas por funciones normales. Estudialo sin copiarlo.

```
from tkinter import *
from tkinter import ttk

def trata_evento(tipo) :
    if tipo == 1 :
        texto = 'Moved mouse inside'
        color = '#96DED1'
    elif tipo == 2 :
        texto = 'Moved mouse outside'
        color = '#FFC300'
    elif tipo == 3 :
        texto = 'Clicked left mouse button'
        color = '#87CEEB'
    l.configure(text=texto, bg=color)

root = Tk()
root.geometry('300x300')
l =Label(root, text="Zona cero...", bg="green", fg="white",
font=("Verdana",18))
l.grid()
```

Contiua ...

```
l.bind('<Enter>', lambda e: trata_evento(1))
l.bind('<Leave>', lambda e: trata_evento(2))
l.bind('<1>',     lambda e: trata_evento(3))
l.bind('<Double-1>', lambda e: l.configure(text='Double clicked'))
l.bind('<B3-Motion>',
               lambda e: l.configure(text='right button drag to
%d,%d' % (e.x, e.y)))

root.mainloop()
```

<B1-Motion>

The mouse is moved, with mouse button 1 being held down (use B2 for the middle button, B3 for the right button). The current position of the mouse pointer is provided in the **x** and **y** members of the event object passed to the callback.

<ButtonRelease-1>

Button 1 was released. The current position of the mouse pointer is provided in the **x** and **y** members of the event object passed to the callback.

<Double-Button-1>

Button 1 was double clicked. You can use **Double** or **Triple** as prefixes. Note that if you bind to both a single click (<**Button-1**>) and a double click, both bindings will be called.

<Enter>

The mouse pointer entered the widget (this event doesn't mean that the user pressed the **Enter** key!).

<Leave>

The mouse pointer left the widget.

<FocusIn>

Keyboard focus was moved to this widget, or to a child of this widget.

https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

<Return>
The user pressed the Enter key. You can bind to virtually all keys on the keyboard. For an ordinary 102-key PC-style keyboard, the special keys are **Cancel** (the Break key), **BackSpace**, **Tab**, **Return**(the Enter key), **Shift_L** (any Shift key), **Control_L** (any Control key), **Alt_L** (any Alt key), **Pause**, **Caps_Lock**, **Escape**, **Prior** (Page Up), **Next** (Page Down), **End**, **Home**, **Left**, **Up**, **Right**, **Down**, **Print**, **Insert**, **Delete**, **F1**, **F2**, **F3**, **F4**, **F5**, **F6**, **F7**, **F8**, **F9**, **F10**, **F11**, **F12**, **Num_Lock**, and **Scroll_Lock**.

<Key>
The user pressed any key. The key is provided in the **char** member of the event object passed to the callback (this is an empty string for special keys).

a
The user typed an "a". Most printable characters can be used as is. The exceptions are space (**<space>**) and less than (**<less>**). Note that **1** is a keyboard binding, while **<1>** is a button binding.

<Shift-Up>
The user pressed the Up arrow, while holding the Shift key pressed. You can use prefixes like **Alt**, **Shift**, and **Control**.

<Configure>
The widget changed size (or location, on some platforms). The new size is provided in the **width** and **height** attributes of the event object passed to the callback.

Ejercicio : Puedes encontrar más información en :

Vinculando eventos :

https://www.pythontutorial.net/tkinter/tkinter-event-binding/

Introduction to the Tkinter event binding

Assigning a function to an event of a widget is called **event binding**. When the event occurs, the assigned function is invoked automatically.

In the previous tutorial, you learned how to bind a function to an event of a widget via the `command` option. However, not all Tkinter widgets support the `command` option.

Therefore, Tkinter provides you with an alternative way for event binding via the `bind()` method. The following shows the general syntax of the `bind()` method:

```
widget.bind(event, handler, add=None)
```

Funciones
Mas a fondo


```
>>> def super_funcion(*args,**kwargs):
...     total = 0
...     for arg in args:
...         total += arg
...     print "sumatorio => ", total
...     for kwarg in kwargs:
...         print kwarg, "=>", kwargs[kwarg]
...
>>> super_funcion(50, -1, 1.56, 10, 20, 300, cms="Plone", edad=38)
sumatorio =>  380.56
edad => 38
cms => Plone
```

4 Funciones lambda

En *Python*, una función Lambda se refiere a una pequeña función anónima. Las llamamos “funciones anónimas” porque técnicamente se definen sin nombre.

En su lugar, las funciones Lambda se crean como una línea que ejecuta una sola expresión. Este tipo de funciones pueden tomar cualquier número de argumentos, pero solo pueden tener una expresión.

```
lambda param1, param2 : expresión_de_retorno
```

```
#Aquí tenemos una función creada para sumar.

def suma(x,y):
   return(x + y)

#Aquí tenemos una función Lambda que también suma.
lambda x,y : x + y


#Para poder utilizarla necesitamos guardarla en una variable.
suma_dos = lambda x,y : x + y
```

```python
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(3)) # Resultado: 9

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
  return num ** 2
print(square(5)) # Resultado: 25
```

Ejercicio : Realiza los ejercicios de esta web.

https://docs.hektorprofe.net/python/funcionalidades-avanzadas/funciones-lambda/

5 Funciones a fondo parámetros por valor, y refer

Paso por valor y referencia

Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

- **Paso por valor**: Se crea una copia local de la variable dentro de la función.
- **Paso por referencia**: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

- **Los tipos simples se pasan por valor**: Enteros, flotantes, cadenas, lógicos...
- **Los tipos compuestos se pasan por referencia**: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(numero):
    numero *= 2

n = 10
doblar_valor(n)
print(n)
```

```
10
```

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2

ns = [10,50,100]
doblar_valores(ns)
print(ns)
```

```
[20, 100, 200]
```

6 Argumentos indeterminados

Quizá en alguna ocasión no sabemos de antemano cuantos elementos vamos a enviar a una función. En estos casos podemos utilizar los parámetros indeterminados por posición y por nombre

https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-indeterminados/

Por posición

Para recibir un número indeterminado de parámetros por posición, debemos crear una lista dinámica de argumentos (una tupla en realidad) definiendo el parámetro con un asterisco:

```
def indeterminados_posicion(*args):
    for arg in args:
        print(arg)

indeterminados_posicion(5,"Hola",[1,2,3,4,5])
```

```
5
Hola
[1, 2, 3, 4, 5]
```

Por nombre

Para recibir un número indeterminado de parámetros por nombre (clave-valor o en inglés *keyword args*), debemos crear un diccionario dinámico de argumentos definiendo el parámetro con dos asteriscos:

```
def indeterminados_nombre(**kwargs):
    print(kwargs)

indeterminados_nombre(n=5, c="Hola", l=[1,2,3,4,5])
```

```
{'n': 5, 'c': 'Hola', 'l': [1, 2, 3, 4, 5]}
```

```
def indeterminados_nombre(**kwargs):
    for kwarg in kwargs:
        print(kwarg, "=>", kwargs[kwarg])

indeterminados_nombre(n=5, c="Hola", l=[1,2,3,4,5])
```

```
n => 5
c => Hola
l => [1, 2, 3, 4, 5]
```