



---

# AMPLIACIÓN DE CONOCIMIENTOS PYTHON 3

---

DA2D1E



**JUAN JOSÉ LÓPEZ VEGA**  
**ALEXANDRU BLAGA**

## 1. Introducción

Para esta actividad hemos decidido hacer un proyecto general que recopile todas las nuevas tecnologías que hemos investigado.

Hemos creado un conversor de monedas ( euro, dólar, bitcoin y libra esterlina ) que actualiza sus datos en tiempo real y usando una interfaz gráfica en Tkinter

## 2. División del proyecto

Durante esta actividad hemos tenido el siguiente reparto:

Juan José López Vega → Tkinter, instalación de dependencias, funciones matemáticas, ejemplos y documentación

Alexandru Blaga → Manejo de ficheros, instalación de módulos externos, uso de clases en Python, uso de Selenium.

Nuestro trabajo consta de un proyecto grande (conversor de divisas) y una recopilación de ejemplos sobre las tecnologías utilizadas

### ¡ Importante !

- Para probar el proyecto hay que acceder al archivo App.py y ejecutarlo. No se necesita hacer nada más, el resto es automático
- Es imprescindible tener acceso a internet ya que la información se saca de Google.com
- La aplicación tarda un poco en extraer los datos e instalar las dependencias si es la primera vez que se ejecuta, rogamos paciencia, la espera no será nunca mayor a 1 minuto.

## 3. Tkinter, interfaces gráficas con Python 3

### ¿Qué es Tkinter?

- Tkinter es una librería de Python 3 que nos permite desarrollar programas con interfaces gráficas.

### ¿Qué aporta Tkinter?

- Permite crear aplicaciones de escritorio, con él damos un gran salto a la hora de programar, se acabó ejecutar en consola cada uno de nuestros programas
- Permite la interacción programa-usuario

### Documentación Tkinter

#### a) Ventanas en Tkinter:

Una ventana es un área visual rectangular, la cual contendrá todos nuestros iconos, botones, cuadros de entrada de texto, entre otros. Existen dos tipos: las ventanas de aplicación, que inician y finalizan las aplicaciones gráficas, y las ventanas de dialogo, que permiten la comunicación simple con el usuario, ambas formando la interfaz de usuario.

La ventana va a ser la unidad básica en la que nos vamos a basar para nuestro proyecto

## b) Importación de módulos:

La importación y inicialización es realmente sencilla:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
```

## c) Propiedades de la ventana

En Tkinter nada más declaremos nuestra ventana tenemos diferentes métodos para darle propiedades al que solo se lo tenemos que aplicar a nuestro objeto ventana

- title(): Cambiar el título de la ventana
- mainloop(): es el método principal sin la ventana no se mostrará ,no podemos olvidar ponerlo
- geometry() : configura el tamaño de la ventana
- iconbitmap(): cambia el icono, introduciendo como parámetro la ruta de la imagen relativa de la imagen
- resizable(boolean,boolean): nos permite redimensionar nuestra ventana. El primer parámetro es el eje x y el segundo y, si no queremos que se pueda redimensionar debemos escribir en ambos False.

## d) Widgets

Son los diferentes elementos que irán alojados en nuestra ventana, algunos de los que hemos usado son : Label, Entry ,ComboBox y Button pero también hemos adjuntado ejemplos de RadioButtons y Text

- **Label:** Es una etiqueta de texto donde podremos mostrar información, similar a un <p> en html

```
label = ttk.Label(  
    root,  
    text='A Label with the Helvetica font',  
    font=("Helvetica", 14))  
  
label.pack(ipadx=10, ipady=10)
```

Para crear un Label nos tenemos que crear un objeto y pasarle diferentes atributos

- Contexto : donde lo vamos a implementar, en este caso en la ventana root
- Texto : el contenido que queremos incluir
- Font : una tupla donde incluiremos la fuente y el tamaño

Por último tendremos que usar .pack() o .place() en el elemento para empaquetarlo y decirle las dimensiones que queremos para él , este paso se va a repetir en el resto de Widgets

- **Button:** Como su propio nombre indica es un elemento gráfico que va a recibir un evento de click y va a ejecutar una acción que le vamos a suministrar mediante una función.

```
exit_button = ttk.Button(  
    root,  
    text='Exit',  
    command=lambda: root.quit()  
)
```

En este sencillo ejemplo se ve como se declara un botón como en el label le pasamos el contexto y el texto que queremos añadir.

En particular tenemos el atributo “command” donde le tenemos que pasar la función que queremos que se ejecute cuando pulsamos el botón, en este caso usamos una lambda pero también podemos definirla en otro sitio y llamarla como se muestra ahora

```
def callback():  
    # do something  
  
    ttk.Button(  
        root,  
        text="Demo Button",  
        command=callback  
    )
```

\*\*\* No olvidar hacer .pack() a Button ya que si no, no funcionará como se mencionó antes en el apartado de label \*\*\*

- **Entry:** Este Widget nos permitirá introducir texto y capturarlo para poder trabajar con el mediante código , similar a un <input> de html o un EditText en Android

El primer paso que tenemos que hacer es declarar de manera especial las variables que van a contener información del Entry :

```
email = tk.StringVar()
```

Como siguiente paso tenemos que crearnos nuestro Entry y asociar el valor que contiene al de nuestra variable

```
email_entry = ttk.Entry(signin, textvariable=email)
```

- **ComboBox:** Es el clásico desplegable que te permite elegir una opción entre varias
  - El primer paso es crear una colección con los valores que queremos incluir en el caso del ejemplo es una tupla
  - Nos creamos un Label (opcional ) para indicar al usuario que tiene que seleccionar uno
  - Creamos el combo box , una variable para guardar el dado como en Entry y las asociamos
  - Le pasamos nuestra tupla con los valores y lo empaquetamos
  - Además le hacemos un bind que es una propiedad que nos permite gestionar eventos, en el ejemplo le decimos que cuando sea seleccionado llame a la función month\_changed

```
def month_changed(event):  
    msg = f'You selected {month_cb.get()}!'  
    showinfo(title='Result', message=msg)  
  
# month of year  
months = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',  
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')  
  
label = ttk.Label(text="Please select a month:")  
label.pack(fill='x', padx=5, pady=5)  
  
# create a combobox  
selected_month = tk.StringVar()  
  
month_cb = ttk.Combobox(root, textvariable=selected_month)  
month_cb['values'] = months  
month_cb['state'] = 'readonly' # normal  
month_cb.pack(fill='x', padx=5, pady=5)  
  
month_cb.bind('<<ComboboxSelected>>', month_changed)
```

## 4. Manejo de ficheros con JSON

### ¿Qué es JSON?

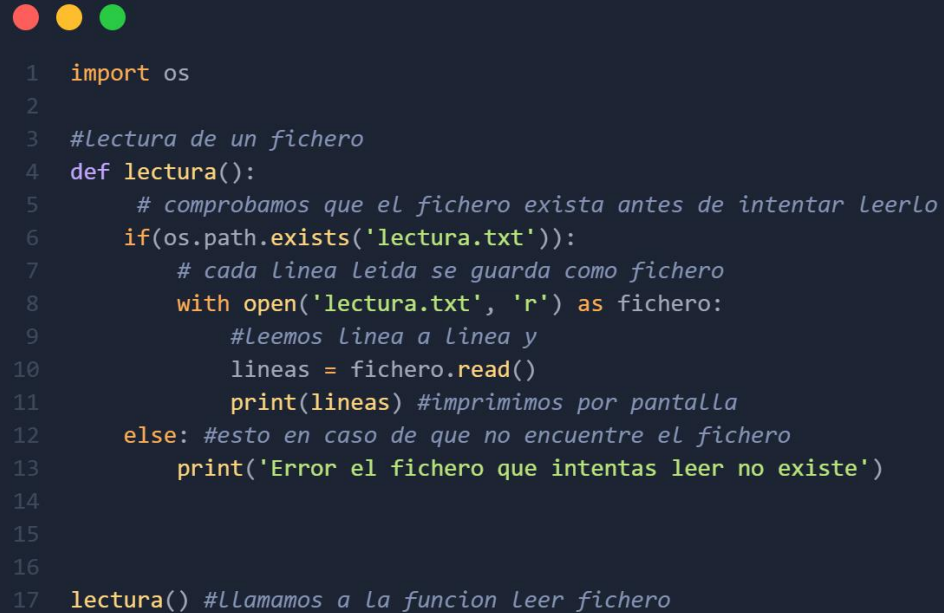
JavaScript Object Model , es una notación en JavaScript ampliamente utilizada en el mundo web y en base de datos para transportar información usando objetos en JavaScript

### ¿Qué nos aporta?

Nos permite guardar información sobre estados, variables en ficheros creados con esta notación para luego usarlo en el momento que nos convenga

Tanto para la lectura como para la escritura hemos usado el módulo propio de Python OS para poder comprobar si el fichero al cual apunta nuestro programa existe o no.

A continuación adjuntamos las capturas de lectura y escritura que explica el uso de este módulo y el de abrir y escribir ficheros.

A screenshot of a code editor window with a dark background and light-colored text. The code is written in Python and is numbered from 1 to 17 on the left side. The code defines a function named 'lectura()' that checks if a file named 'lectura.txt' exists. If it does, it opens the file in read mode ('r') and reads its contents, printing them to the screen. If the file does not exist, it prints an error message. The function is then called at the bottom of the code block.

```
1  import os
2
3  #lectura de un fichero
4  def lectura():
5      # comprobamos que el fichero exista antes de intentar leerlo
6      if(os.path.exists('lectura.txt')):
7          # cada línea leída se guarda como fichero
8          with open('lectura.txt', 'r') as fichero:
9              #leemos línea a línea y
10             líneas = fichero.read()
11             print(líneas) #imprimimos por pantalla
12     else: #esto en caso de que no encuentre el fichero
13         print('Error el fichero que intentas leer no existe')
14
15
16
17  lectura() #llamamos a la función leer fichero
```

```
1  # importamos el modulo os de sistema para poder comprobar si el fichero a escribir ya existe
2  import os
3
4
5  def escritura(texto): # funcion que escribe ficheros de texto en el sistema
6      if(os.path.exists('escritura.txt')): #primero comprobamos que el fichero no exista
7          print('fichero existente!') #en caso de existir salta este print
8      else:
9          #aqui escribimos el fichero de tipo texto
10         with open('escritura.txt', 'w') as fichero:
11             fichero.write(texto) #escribe cada linea
12
13
14
15 #texto a guardar en fichero
16 txt = """
17 esto es un texto
18 multi linea para comprobar
19
20 que no solo escribe lineas simples
21 """
22
23 escritura(txt) #llamamos a la funcion pasandole el parametro
```

## 5. Instalación de dependencias (módulos externos).

### ¿Qué son las dependencias?

Son pequeñas automatizaciones de instalación de paquetes

### ¿Qué nos aportan?

Nos permite instalar paquetes al iniciar un programa sin la necesidad de utilizar la consola, nos ofrece sobre todo capacidad de portabilidad de un programa ya que el receptor no necesitará instalar nada

En nuestra práctica hemos hecho uso de módulos externos que hemos tenido que instalar mediante el gestor de paquetes propio de Python (pip).

Hemos desarrollado una función la cual se encarga de descargar todas las dependencias necesarias para que la aplicación funcione sin problemas además de que el primer paso que da es actualizar el gestor de paquetes para evitar problemas.



```
1 import os #necesitamos el modulo os de sistema para poder instalar las dependencias
2
3 #esta funcion se encarga de instalar los paquetes externos necesarios
4 #para la practica
5 def dependencias():
6     #actualizamos el instalador de paquetes (encargado de instalar todos los paquetes que hay a continuacion)
7     os.system('python -m pip install --upgrade pip')
8     #instalamos el paquete selenium (encargado de manejar el driver de google chrome para traer dicha informacion)
9     os.system('pip install selenium')
10    #instalamos el paquete requests (encargado de hacer las peticiones de la pagina de google)
11    os.system('pip install requests')
12    #instalamos el paquete pillow (encargado de abrir las imagenes)
13    os.system('pip install pillow')
14
```

## 6. Clases en Python.

### ¿Qué son las clases?

Una clase es un tipo de dato definido por el usuario, y al crear instancias de una clase hace relación a la creación de objetos de ese tipo. Las clases y los objetos son considerados los principales bloques de desarrollo para Python, el cual es un lenguaje de programación orientado a objetos

### ¿Qué nos aportan?

Con esta implementación nos permite crear objetos con los atributos y funciones que hemos predefinido, es de extrema utilidad en cualquier lenguaje y en Python no iba a ser menos.

En nuestra práctica hemos tenido que utilizar una clase simple para poder mandar unos datos y solucionar un problema el cual pedía datos antes de abrir la interfaz gráfica.

A continuación he hecho un ejemplo básico de una clase en Python explicando cada parte de ella.

```
1  #ejemplo de clases ampliacion de temario
2
3  class Persona: #clase persona
4
5      #atributos de la clase
6      nombre = None
7      apellido = None
8      edad = None
9
10     #constructor con los atributos a rellenar
11     def __init__(self, nombre, apellido, edad):
12         self.nombre = nombre
13         self.apellido = apellido
14         self.edad = edad
15
16     #getters and setters
17     def getNombre(self):
18         return self.nombre
19
20     def setNombre(self, nombre):
21         self.nombre = nombre
22
23     def getApellido(self):
24         return self.apellido
25
26     def setApellido(self, apellido):
27         self.apellido = apellido
28
29     def getEdad(self):
30         return self.edad
31
32     def setEdad(self, edad):
33         self.edad = edad
34
35     #metodos dentro de la clase
36     def mostrarInfo(self):
37         print('Nombre: ', self.nombre, '\nApellido: ', self.apellido, '\nEdad: ', self.edad)
38
39
```

## 7. Configuración y uso de Selenium.

### ¿Qué es Selenium?

Selenium es un proyecto que alberga un abanico de herramientas y librerías que permiten y apoyan la automatización de navegadores web

### ¿Qué nos aporta?

Selenium nos permite obtener datos a tiempo real de la web como puede ser el valor de las divisas en Google.com, para ello debemos configurarlo y capturar esa información como se muestra a continuación y en los ejemplos

```
1 import os # modulo os para comprobar que el driver ya esta descargado
2 import time # modulo time utilizado para hacer un sleep necesario
3 import zipfile # modulo zipfile necesario para descomprimir el driver de selenium
4 import requests # modulo request necesario para hacer la solicitud de descarga del driver de google
5 from selenium import webdriver #desde selenium importamos el webdriver
6 from selenium.webdriver.chrome.options import Options #tambien desde selenium importamos Options
7
8 #options se encarga de configurar de que forma queremos que selenium actue
9 """
10 En este caso hacemos que selenium oculte la ventana de google al consultar los datos
11 y que no muestre todos los eventos por consola que son innecesarios
12 """
13 # Configuración de las opciones del driver de google para usar selenium
14 options = Options()
15 # hacer que el driver trabaje en segundo plano con selenium
16 options.add_argument('--headless') # evitar que abra una ventana de chrome sobre la cual trabajar
17 options.add_argument('log-level=4') # evitar mensajes de advertencia e informacion innecesaria
18
19
20 def dependencia_driver():
21     if(os.path.isfile('chromedriver.exe')): # comprueba si existe o no el driver para no perder tiempo descargandolo
22         print('El driver ya existe, listo para usarse') #te notifica con un print
23
24     else:
25         print('-->¡ Descargando driver...')
26         # Enlace de descarga del driver
27         url = 'https://chromedriver.storage.googleapis.com/89.0.4389.23/chromedriver_win32.zip'
28         # Hacemos la petición y permitimos que nos redirija
29         req = requests.get(url, allow_redirects = True)
30         # Abrimos dicho fichero de la petición y lo escribimos en formato zip con escritura binaria para no perder datos.
31         open('chromedriver_win32.zip', 'wb').write(req.content)
32
33         # Descomprimos el driver utilizando un modulo interno de Python llamado zipfile
34         with zipfile.ZipFile('chromedriver_win32.zip', 'r') as zip_ref:
35             zip_ref.extractall('.')
36         # Una vez extraído borramos el archivo .zip
37         os.remove('chromedriver_win32.zip')
38         print('-->¡ Driver listo para usarse.')
39     #devuelve el driver listo y configurado
40     return webdriver.Chrome('./chromedriver.exe', options=options)
41
```

```
1 import EjemploDependenciasSelenium as driv #usare el ejemplo anterior para una demostracion de selenium
2 import os
3 import time
4
5
6
7 driver = driv.dependencia_driver()
8
9 #hare un ejemplo de tomar el tiempo actual de madrid (tiempo real)
10
11 """ lo primero que debemos hacer es crear una funcion para pasarle el enlace de la pagina
12 del tiempo y capturar los elements html para extraer los datos de dicha pagina y mostrarlos
13 por consola """
14
15
16 def recoger_tiempo():
17     #comprobamos que el driver este ya disponible descargado
18     if(os.path.isfile('chromedriver.exe')):
19         #al driver le pasamos el enlace de la pagina de donde vamos a tomar los datos
20         driver.get('https://www.google.com/search?q=tiempo+madrid&rlz=1C1CHZN_esES935ES935&sxsrf=ALeKk00m80
21 kygBG6kDICupI-63aY8R448Q:1621374652637&ei=vDakYPG6JpT_sAf-koGABg&ooq=tiempo+madrid&gs_lcp=Cgdn3Mtd2l6EAMyDw
22 gAELEDEIMBEEMQRhCAAjIECAAQzICCAAyBQgAELEDmQIABBDmEIADICCAyAggAMgUIABcxAzICCAA6BAGjECc6CggAELEDEIMBEEM6C
23 AgAELEDEIMBOgYIIxAnEBM6BwgAEMKDEEM6BQgAEJIDoggIABcxAxDJA1CqE1jGKGDkmgAcA34AIABmAGIAaEKgEEMTAuNjgBAKABAaoB
24 B2d3cy13aXrAAQE&sclient=gws-wiz&ved=0ahUKewjxiN-hm9TwAhWUP-wKHx5JAGAQ4dUDCA4&uact=5')
25         #marcamos un sleep de dos segundos para que cargue la pagina en segundo plano
26         time.sleep(2)
27         #creamos la variable tiempo en la cual guardaremos el elemento html
28         tiempo = driver.find_element_by_xpath('//*[@id="wob_tm"]').text
29         #dicho elemento ya ha sido capturado y se ha extraido el valor en formato de texto
30         print(type(tiempo)) #devuelve str
31         #ahora podemos mostrar por pantalla el tiempo actual o devolver el valor mediante un return
32         return tiempo
33
34 resultado = recoger_tiempo()
35
36 print(resultado, "Grados celsius")
37
```

## 8. Bibliografía utilizada

- <https://www.w3schools.com/python/>
- <https://docs.python.org/es/3/>
- <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/tutorial-de-selenium-webdriver/>
- <https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>