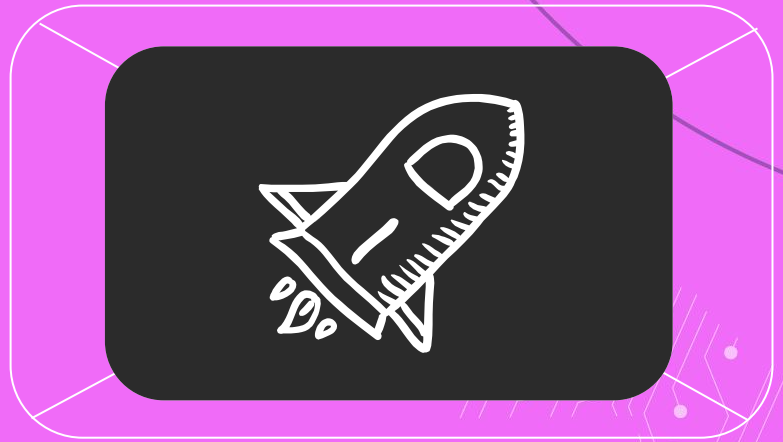
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

Semana 4
¡Bienvenidos!



Temas de hoy

1. Módulos: Importación de módulos estándar y módulos personalizados.
2. Tkinter (Practicando nuestros primeros programas gráficos 😉)

1

Módulos

Ahora que ya viste funciones y conocés cómo funcionan, es momento de darle un uso mayor, y ese uso, es el de “transportarlas”.

Podemos decir que los módulos son como cajas muy grandes, que pueden guardar funciones, variables, etc…

Con esto podemos reutilizar código en un u otro programa ¡Vamos a verlo! 👉

Módulos: módulos estándar y personalizados

Módulos estándar

Va a ocurrir que hacemos una función que nos sirve para un programa, pero luego, haciendo otro programa, nos damos cuenta que esa función que habíamos hecho, también puede servir para el otro programa, por ejemplo, una función de una calculadora.
Con los módulos, no va a ser necesario ir a copiar ese código que ya hicimos y traerlo al nuevo programa. Sólo hay que importarlo al programa que queremos y comenzar a utilizarlo.
Entonces, un módulo es completamente útil para reutilizar código, ahorrar tiempo y sobre todo, ahorrarnos líneas de código. 🤓
Pero ¿qué es un módulo estándar? En uno de los primeros códigos de ejemplo que te dimos (en la primer clase), usamos un módulo estándar, pero aún no sabías mucho del mundo de la programación, así que lo dejamos para explicarlo más adelante, es decir, ahora.

```
1.2.py >
1  import datetime
2
3  def verificar_mayoria_edad():
4      try:
5          anio_nac = int(input("Ingresa tu año de nacimiento: "))
6          anio_actual = datetime.datetime.now().year
7          edad = anio_actual - anio_nac
```

Concretamente en este ejemplo, usamos una función para verificar la edad, pero además, importamos a nuestro programa el módulo estándar "**datetime**", que es un módulo incluído en Python. Este módulo maneja las fechas (lo habrás intuido por su nombre).
Otra cosa a destacar, es que para llamar a un módulo estándar solo basta con usar la palabra reservada ***import*** y luego el nombre del módulo.
Para saber cómo usar sus funciones hay que ver el código del módulo, tal cual vemos en el ejemplo para la variable

Módulos: módulos estándar y personalizados

"anio_actual", y estas funciones la podemos ver en la documentación de Python (https://docs.python.org/3/py-modindex.html)

Un módulo es simplemente un archivo con extensión *.py* que contiene código Python. Por ejemplo, si tenemos que realizar un programa donde necesitamos generar números aleatorios, no sería necesario programar el código para realizar esta función, ya que existe un módulo estándar que lo hace, por lo que, importando este módulo podemos generar números aleatorios y ahorrar código y tiempo.
En el siguiente ejemplo donde te vamos a mostrar varias cosas 🤓

modulos.py > ...

```python
import random

def generador_numeros():
    return random.randint(1, 100)

print('¡Bienvenido al generador de números aleatorios del Info! (números del 1 al 100).')
respuesta = input('¿Querés comenzar? S/N: ')

while respuesta.upper() == 'S':
    numero = generador_numeros()
    print(f'El número generado es: {numero}')
    respuesta = input('¿Querés seguir generando números aleatorios? S/N: ').upper()

print('¡Gracias por jugar con el generador de números del Info!')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
¡Bienvenido al generador de números aleatorios del Info! (números del 1 al 100).
¿Querés comenzar? S/N: s
El número generado es: 46
¿Querés seguir generando números aleatorios? S/N: s
El número generado es: 56
¿Querés seguir generando números aleatorios? S/N: s
El número generado es: 40
¿Querés seguir generando números aleatorios? S/N: n
¡Gracias por jugar con el generador de números del Info!
```

Módulos: módulos estándar y personalizados

Vamos a ir viendo las partes de este ejemplo.

- **Importamos**:
```
1  import random
```
Cómo podés ver, en la primer línea de código importamos el módulo estándar **random** utilizando el comando ***import***.

- **Generamos una función:**
```
3  def generador_numeros():
4      return random.randint(1, 100)
```
Mediante la función que creamos, utilizamos en el bloque de código, la funcionalidad de ***random.randint()*** para generar números aleatorios entre el 1 y el 100.

- **Presentamos el programa:**
```
6  print('¡Bienvenido al generador de números aleatorios del Info! (números del 1 al 100).')
7  respuesta = input('¿Querés comenzar? S/N: ')
```
Damos un mensaje de bienvenida al usuario y luego pedimos una respuesta para iniciar el programa.

- **Generamos un bucle:**
```
9  while respuesta.upper() == 'S':
10     numero = generador_numeros()
11     print(f'El número generado es: {numero}')
12     respuesta = input('¿Querés seguir generando números aleatorios? S/N: ').upper()
```
Mediante el bucle ***while*** generemos un bloque de código donde el usuario controla la cantidad de veces que se generarán números aleatorios. Podés observar que mediante la invocación de la función que creamos anteriormente, guardamos el número generado en la variable *numero* y lo mostramos por pantalla.
Luego solicitamos nuevamente una respuesta del usuario para continuar o no en el programa. Algo a destacar es que, al inicio del bucle, a la variable *respuesta* le aplicamos el método *upper()* para igualar la respuesta del usuario a mayúscula, y luego, cuando volvemos a hacer la pregunta, usamos el método *upper()* al final del *input*.
Esto lo hicimos así para mostrarte que se puede usar de distintas formas este método.

Módulos: módulos estándar y personalizados

Módulos personalizados

En el ejemplo anterior pudimos ver un ejemplo para importar un módulo estándar (*random*), sin embargo, va a haber veces que vamos a necesitar hacer nuestra propia función y convertirla en un módulo para ser usada en otros programas.
Te recordamos que un módulo es un archivo con extensión *.py* con código Python, por lo que para hacer nuestro propio módulo, primero debemos crear la función que necesitamos y guardar ese archivo.
¿Vemos un ejemplo?🧐

biblioteca.py > ...

```python
def crear_libro(titulo, autor, anio):
    return {'titulo': titulo, 'autor': autor, 'año': anio}

def agregar_libro(biblioteca, libro):
    biblioteca.append(libro)

def buscar_libro(biblioteca, titulo):
    for libro in biblioteca:
        if libro['titulo'] == titulo:
            return libro
    return None

def mostrar_catalogo(biblioteca):
    for libro in biblioteca:
        print(f'Título: {libro['titulo']}, Autor: {libro['autor']}, Año: {libro['año']}')

def funcion_de_muestra():
    pass
```

Primero, creamos nuestro módulo el cual se basa en una biblioteca.
Fijate que estamos integrando todo lo aprendido hasta acá. Estamos creando *Funciones* para: Crear un libro, Agregar un libro, Buscar un libro y Mostrar un catálogo. Estos datos los vamos guardando y mostrando con en *Listas* y *Diccionarios*. Además, estamos usando *Condicionales* y *Bucles* (además de mostrar por consola con *f-string*).

Módulos: módulos estándar y personalizados

Con el módulo creado, podemos generar el programa de biblioteca, por lo que ¡vamos a hacerlo! 😃

biblioteca_info.py > ...

```python
from biblioteca import crear_libro, agregar_libro, buscar_libro, mostrar_catalogo

biblioteca_del_info = []

libro1 = crear_libro('Lógica de programación', 'Omar Iván Trejos Buriticá', 2017)
libro2 = crear_libro('Lógica de programación orientada a objetos', 'Efrain Oviedo Regino', 2015)

agregar_libro(biblioteca_del_info, libro1)
agregar_libro(biblioteca_del_info, libro2)

mostrar_catalogo(biblioteca_del_info)

libro_buscado = buscar_libro(biblioteca_del_info, 'Lógica de programación')
print(libro_buscado)
```

Fijate como ahora estamos importando nuestro propio módulo personalizado en la primer línea de código, ¡pero acá estamos haciendo algo nuevo!

Estamos usando una nueva palabra clave o comando: *from*

Con este comando, lo que hacemos es decirle a Python que “desde” el módulo *biblioteca* vamos a importar las funciones *crear_libro, agregar_libro, buscar_libro, mostrar_catalogo* las cuales representan a todas las funciones que componen nuestro módulo.

¿Por qué lo hacemos así? Para mostrarte que no necesariamente, hay que importar todas las funciones que tiene un módulo. Podemos importar solamente lo que necesitamos, y por eso creamos la función *funcion_de_prueba()* que no tiene nada en su bloque de código, pero lo hicimos para que veas que no estamos llamando a esta función al importar el módulo *biblioteca* a nuestro programa.

¿Y qué diferencia hay haciendo esto? Fácil, la legibilidad del código.

Suele abrirse el debate sobre el rendimiento del programa en estos casos, pero la realidad es que, en general, importar partes del módulo o todo el módulo completo, no repercute en el rendimiento, pero lo repetimos, en general.

Pero vamos a abordar el punto de la legibilidad.

Módulos: módulos estándar y personalizados

¿Importar las funciones necesarias o importar el módulo completo?

from *módulo* ***import*** *funciones o clases* : La mayoría de las veces no vamos a usar todas las funcionalidades que ofrece un módulo, por lo que no es necesario importarlo por completo.
Con esto podemos evitarnos conflictos de nombres con otras *funciones* o *clases* (no te preocupes que *clases* lo veremos un poco más adelante), o incluso, variables que pueda traer el módulo y pueda llegar a haber en el código del programa en el que se está trabajando.
De esta forma logramos un código más conciso.

import *módulo* : Importamos un módulo completo cuando necesitamos acceder a múltiples elementos del módulo. Además, con esto también podemos mantener claro de dónde proviene cada elemento.

Otras formas de importar: también tenemos otras formas de importar donde si utilizamos:

- ***from*** *módulo* ***import*** ***** : Importamos todos los elementos públicos del módulo directamente en el espacio de nombres actual, por lo que existe el riesgo de que si hay elementos con el mismo nombre en tu código, se produzca un conflicto y se sobreescriba alguno de ellos. Esto puede arrojar errores difíciles de detectar, sobre todo cuando estás empezando en la programación. Por lo que una buena práctica es evitar esta forma de importar, sobre todo en proyectos grandes o cuando se trabaja en equipo.
  Ejemplo: ***from*** *math* ***import ****

- ***from*** *módulo* ***import as*** *alias* : Esta forma de importar la usamos cuando necesitamos importar varios elementos de un módulo pero queremos evitar escribir nombres largos.

Módulos: módulos estándar y personalizados

Lo que nos brinda un alias, es un nombre alternativo que le asignamos a un módulo o un elemento dentro de un módulo al momento de importarlo, lo cual resulta muy útil cuando los nombres son muy largos y queremos simplificar el código y hacerlo más legible. Además, podemos evitar que se sobreescriban nombres de variables, funciones o clases existentes en nuestro código.
Otra cosa muy importante, es que hay ciertas convenciones muy utilizadas en la comunidad de Python donde para módulos como *Pandas* o *NumPy* se suele utilizar alias. Por ejemplo:
import *pandas* ***as*** *pd*
import *numpy* ***as*** *np*

Entonces ¿cuándo usar cada uno?

- **import módulo :** Ideal para la mayoría de los casos, especialmente en proyectos grandes y complejos.

- **from módulo import * :** Utilízalo con precaución y solo cuando estés seguro de que no habrá conflictos de nombres. Puede ser útil para módulos pequeños y bien conocidos.

- **from módulo import elemento :** Útil cuando solo necesitas algunos elementos específicos de un módulo y quieres evitar importar todo.

- **from módulo import elemento as alias :** Útil para nombres largos, conflictos de nombres o para establecer convenciones en tu proyecto.

Módulos: Tkinter

Importando un módulo para crear interfaces gráficas🤩

Sabemos que solo ver resultados por la consola de comandos, puede resultar un poco tedioso 😪 así que para darle un poquito más de “brillo” a esto, vamos a ver algunos gráficos creados ¡con lo que programamos!

Tkinter es un módulo de Python ¡muy popular! con el que se pueden crear interfaces gráficas de usuario (GUI).
Veamos cómo importarlo y un ejemplo simple:

```
hola_mundo_tkinter.py > ...
1   import tkinter as tk
```

Ejemplo simple: Ventana básica

```
hola_mundo_tkinter.py > ...
1   import tkinter as tk
2
3   # Creamos una ventana principal
4   ventana = tk.Tk()
5   ventana.title('Mi primera ventana con Tkinter')
6   ventana.geometry("400x300")  # Ancho x Alto en píxeles
7
8   # Creamos un label (como en etapa 1 en Formularios ¿lo recordás?)
9   etiqueta = tk.Label(ventana, text='¡Hola, mundo!', font=('Arial', 20))
10  etiqueta.pack()
11
12  # Iniciamos el bucle principal de nuestra aplicación
13  ventana.mainloop()
```

Y si ejecutamos 🤩:

Módulos: Tkinter

¡Ahora si! Dejamos de ver solamente resultados por consola y podemos ver algo gráfico 😄pero para que comprendas un poco mejor, vamos a explicarte las partes del código este ejemplo:

Importar Tkinter: Importamos el módulo Tkinter y lo asignamos al alias **tk**.

Crear la ventana principal: Creamos una instancia de la clase **Tk** y la asignamos a la variable *ventana*. Esta será la ventana principal de nuestra aplicación.

Establecer el título: Usamos el método *title()* para asignar un título a la ventana.

Crear un label: Creamos un objeto *Label* que mostrará el texto "¡Hola, mundo!". El primer argumento es la variable *ventana*, el segundo argumento es el texto que queremos mostrar y, el tercero es el tamaño de la fuente.

Empaquetar el label: Usamos el método *pack()* para colocar el *label* dentro de la ventana. Hay otros métodos como *grid()* y *place()* para organizar los elementos de una forma más precisa.

Iniciar el bucle principal: *mainloop()* inicia el bucle principal de la aplicación, que se encarga de manejar los eventos del usuario y mantener la ventana abierta hasta que se cierre.

Con este ejemplo básico como punto de partida, podés comenzar a crear interfaces gráficas más complejas y personalizadas😉

Te invitamos a ver la documentación oficial de Tkinter: https://docs.python.org/es/3/library/tkinter.html

Módulos: Tkinter - Un poco más de práctica

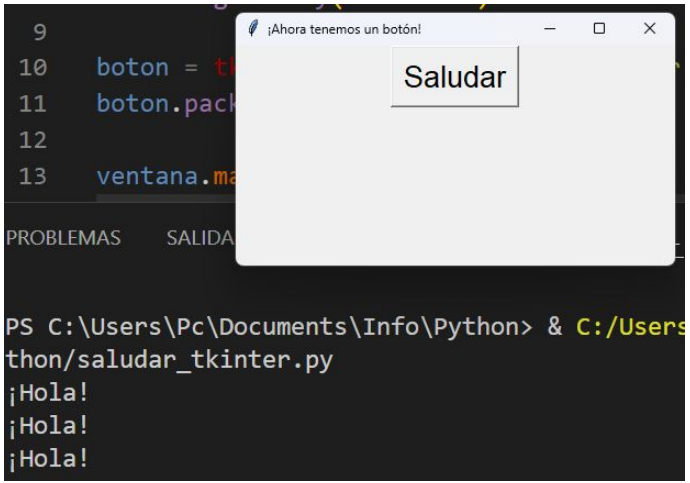
¿Te parece si creamos un botón?

¡Esperamos que la respuesta sea sí! Por lo que vamos con el ejemplo:

saludar_tkinter.py > ...

```python
import tkinter as tk

def saludar():
    print('¡Hola!')

ventana = tk.Tk()
ventana.title('¡Ahora tenemos un botón!')
ventana.geometry('400x200')

boton = tk.Button(ventana, text='Saludar', font=('Arial', 20), command=saludar)
boton.pack()

ventana.mainloop()
```

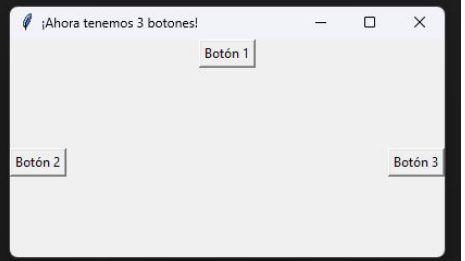
Ejecutamos:



Módulos: Tkinter - Un poco más de práctica

¿Y si son 3 botones?

```python
import tkinter as tk

ventana = tk.Tk()
ventana.title('¡Ahora tenemos 3 botones!')
ventana.geometry('400x200')

boton1 = tk.Button(ventana, text="Botón 1")
boton2 = tk.Button(ventana, text="Botón 2")
boton3 = tk.Button(ventana, text="Botón 3")

boton1.pack(side="top")
boton2.pack(side="left")
boton3.pack(side="right")

ventana.mainloop()
```



Mini proyectos ¡Muy pronto!

Ahora que ya te estás introduciendo al mundo gráfico con Tkinter y vimos algunos ejemplos, seguramente querés ¡seguir creando mucho más! 😃 Pero para hacerlo un poco más dinámico, dentro de muy poco te vamos a dar unos miniproyectos para que los hagas en grupo.

Pero aún hay que tener ¡paciencia! Ya que primero te vamos a hablar de otro tema en la siguiente clase, para que la gestión de los miniproyectos sea de lo más óptima y eficiente posible y para eso, hay que tener ciertas metodologías a seguir y optimizar completamente todo.

🫡

¡Nos vemos En la próxima clase!