«Talento Tech»

Iniciación a la

Programación con Python

Clase 11

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 11 | Módulos

Temario:

- Organización del código en módulos.
- Importación de módulos propios y externos.
- Uso de módulos estándar (random, datetime).
- Introducción a módulos de terceros (colorama).

Objetivos de la Clase

En esta clase, aprenderás a organizar y estructurar tu código mediante el uso de módulos en Python. Aprenderás a dividir tu código en archivos y componentes reutilizables, mejorando la mantenibilidad y facilitando la colaboración en proyectos.

Explorarás también la poderosa funcionalidad de los módulos estándar, como random y datetime, y cómo aprovechar sus herramientas para resolver problemas comunes de manera eficiente. Además, darás tus primeros pasos en el uso de módulos de terceros, como colorama, lo que te permitirá enriquecer tus aplicaciones con funcionalidades adicionales.

Al finalizar la clase, serás capaz de:

- Importar y utilizar módulos propios y externos en tus proyectos.
- Identificar cómo los módulos contribuyen a escribir un código más limpio y organizado.
- Incorporar librerías estándar y de terceros para expandir las capacidades de tus programas.

Este conocimiento te brindará una base sólida para construir soluciones modulares y escalables, preparándote para enfrentar nuevos desafíos en el desarrollo de software.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Bienvenido a tu jornada en TalentoLab 🎉

Talento Lab

El día comienza con la energía habitual en **TalentoLab**. Mariana te llama a su oficina apenas llegás, con su característico entusiasmo. "Hoy es un gran día para profundizar en la organización del código". Te explica qué vas a aprender a trabajar con módulos, una herramienta esencial para dividir y organizar el código en partes más manejables y reutilizables.

Luis, que pasa casualmente por la oficina, se suma a la charla. "¿Te acordás del proyecto del TFI? Imaginate lo útil que sería dividir ese código en partes", te dice con una sonrisa. “*O potenciarlo utilizando funciones o código escrito y probado por otras personas*”, agrega.

Para empezar el día, te asignan una tarea sencilla: explorar cómo importar y usar módulos ya disponibles en Python, como **random** y **datetime**. Además, te adelantan que vas a tener la oportunidad de probar un módulo de terceros, **colorama**, para darle un toque de color a tus aplicaciones de consola. "Es una herramienta genial para que tus programas sean más atractivos visualmente", comenta Luis. **Hoy, el enfoque está en aprender a organizar, reutilizar y ampliar tu código con la ayuda de los módulos en Python.**

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción a los módulos.

Un módulo es un archivo que contiene definiciones y declaraciones de funciones, clases o variables, diseñado para ser reutilizado en otros programas. Los módulos permiten organizar el código en componentes separados y reutilizables, lo que simplifica la lectura, el mantenimiento y la escalabilidad de los proyectos. Python incluye una gran cantidad de **módulos estándar** que cubren tareas comunes, desde matemáticas y manejo de fechas hasta la generación de números aleatorios y la interacción con archivos.

Por ejemplo, cuando escribís un programa que necesita realizar cálculos matemáticos avanzados, podés importar el módulo **math** en lugar de escribir todas las funciones matemáticas desde cero. Si tu programa requiere generar números aleatorios, el módulo **random** ofrece una solución rápida y eficiente.

¿Qué es un módulo y para qué sirve?

Los módulos sirven para dividir un programa en partes más pequeñas y manejables, mejorando la estructura del código. Además, permiten reutilizar funcionalidades en múltiples proyectos sin necesidad de reescribirlas. Por ejemplo, un módulo creado para calcular impuestos puede ser usado en diferentes aplicaciones relacionadas con contabilidad o ventas.

Además, los módulos ayudan a evitar problemas con nombres de variables y funciones al encapsularlos en su propio espacio de nombres. Esto es especialmente útil en proyectos grandes, donde diferentes partes del código podrían necesitar definiciones similares.

Ventajas de usar módulos en Python

Utilizar módulos en tus proyectos ofrece varias ventajas:

1. **Reutilización de código:** Podés escribir funciones y clases una vez en un módulo y usarlas en múltiples programas, ahorrando tiempo y esfuerzo.
2. **Organización:** Los módulos dividen el código en bloques lógicos, haciendo que sea más fácil entender qué hace cada parte de tu programa.
3. **Mantenibilidad:** Es más sencillo actualizar o corregir errores en un módulo específico sin afectar otras partes del código.

4. **Escalabilidad:** En proyectos grandes, los módulos permiten trabajar en equipo de manera más eficiente, ya que cada integrante puede enfocarse en una parte del proyecto.
5. **Acceso a herramientas listas para usar:** Python ofrece una amplia biblioteca estándar con módulos que resuelven problemas comunes, y también podés instalar módulos de terceros para ampliar las capacidades de tu proyecto.

Por ejemplo, si tu aplicación necesita calcular fechas futuras basadas en datos ingresados por usuarios, podés usar el módulo **datetime**. Esto no solo te ahorra escribir el código desde cero, sino que también asegura que estés usando herramientas probadas y optimizadas.

¿Cómo dividir tu código en diferentes archivos?

Dividir el código en diferentes archivos es una práctica fundamental en el desarrollo de software que ayuda a mantener la organización, la claridad y la escalabilidad de tus proyectos. En Python, cada archivo con la extensión **.py** se considera un módulo y puede contener funciones, clases y variables que podés reutilizar en otros archivos de tu proyecto.

Para dividir tu código en diferentes archivos, seguí estos pasos básicos:

1. **Identificá las partes del código que pueden agruparse:** Analizá tu programa y determiná qué funcionalidades o secciones podrían organizarse en módulos. Por ejemplo, podrías tener un módulo para funciones matemáticas, otro para manejo de bases de datos y uno más para la interfaz de usuario.
2. **Creá archivos `.py` para cada módulo:** Guardá las funciones, clases o variables relacionadas en un archivo separado con un nombre que describa claramente su contenido. Por ejemplo, un archivo llamado **calculadora.py** podría contener funciones para sumar, restar, multiplicar y dividir.
3. **Importá el módulo en tu programa principal:** Usá la palabra clave **import** para incluir el módulo en el archivo principal de tu programa. Esto te permitirá acceder a las funciones y variables definidas en otros archivos.

Por ejemplo, supongamos que tenés un archivo principal **programa.py** y querés dividir una sección del código en un módulo llamado **utilidades.py**. En el archivo **utilidades.py**, podrías definir una función como esta:

```
# utilidades.py
def saludar(nombre):
   print(f"Hola, {nombre}. ¡Bienvenido!")
```

En tu archivo principal **programa.py**, simplemente importás el módulo para usar esa función:

```
# programa.py
import utilidades
utilidades.saludar("Mariana")
```

El código queda más organizado, y cualquier cambio en las funciones definidas en **utilidades.py** se reflejará automáticamente en los programas que lo utilicen. Pero antes de seguir, veamos qué es lo que hace y cómo se utiliza **import**.

La orden import: Sintaxis, uso y detalles.

La orden **import** es fundamental para aprovechar la modularidad del lenguaje. Permite incorporar el contenido de otros módulos a tu programa actual, haciendo posible reutilizar código y acceder a funciones, clases y variables definidas en otros lugares. La sintaxis básica de **import** es simple:

```
import nombre_modulo
```

En este caso, **nombre_modulo** es el nombre del archivo **.py** que querés importar, sin incluir la extensión **.py**. Una vez importado, podés acceder a las funciones, clases o variables del módulo utilizando la notación **nombre_modulo.elemento**.

Por ejemplo:

```
# utilidades.py
def saludar(nombre):
   print(f"Hola, {nombre}. ¡Bienvenido!")
```

```
# programa.py
import utilidades
utilidades.saludar("Luis")
```

Importar elementos específicos de un módulo
Si sólo necesitás ciertas funciones, clases o variables de un módulo, podés importarlas directamente usando la sintaxis:

```
from nombre_modulo import elemento1, elemento2
```

Por ejemplo:

```
from utilidades import saludar
saludar("Mariana")  # No es necesario usar utilidades.saludar
```

Esto simplifica el código, pero tené cuidado con los posibles conflictos de nombres si usás esta técnica con varios módulos.

Importar con un alias
Para facilitar la lectura o resolver conflictos de nombres, podés usar un alias con la palabra clave **as**:

```
import utilidades as ut
ut.saludar("Diego")
```

Esto es útil cuando el nombre del módulo es largo o si querés estandarizar los nombres en un equipo de desarrollo.

Importar todo el contenido de un módulo
También es posible importar todo el contenido de un módulo usando un asterisco *:

```
from utilidades import *
```

Aunque esta técnica es conveniente en algunos casos, no se recomienda porque puede generar conflictos **de** nombres y hacer que el código sea más difícil de entender.

¿Cómo funciona import internamente?

Cuando usás **import**, Python busca el módulo en los directorios especificados en la variable sys.path. Por defecto, incluye:

- El directorio del script que se está ejecutando.
- Directorios estándar del sistema donde están instaladas las librerías de Python.
- Directorios adicionales especificados por vos (si los agregás manualmente).

Si Python no encuentra el módulo, lanzará un error (**ModuleNotFoundError**).

Ventajas del uso de import

- Reutilización de código: Podés usar el mismo módulo en múltiples programas.
- Organización: Te permite dividir el código en archivos más pequeños y manejables.
- Colaboración: Facilita que varios desarrolladores trabajen en diferentes módulos de un proyecto.

Con la orden **import**, estás desbloqueando una de las herramientas más poderosas de Python para trabajar de manera modular, eficiente y escalable.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Módulos estándar.

Los **módulos estándar** son una parte esencial del ecosistema de Python. Estos módulos vienen incluidos por defecto en cada instalación del lenguaje y están diseñados para ofrecer una amplia variedad de herramientas que facilitan la resolución de problemas comunes y optimizan el desarrollo de aplicaciones. Gracias a su disponibilidad inmediata, los módulos estándar se convierten en aliados fundamentales para cualquier programador que utilice Python, ya que no requieren instalación adicional y están listos para usarse mediante la orden **import**.

A diferencia de los módulos creados por el usuario, que son específicos para un proyecto o aplicación, los módulos estándar están diseñados para ser universales y aplicables en una variedad de contextos. Por ejemplo, un módulo estándar como **random** permite generar números aleatorios de manera sencilla, mientras que un módulo que creas vos podría contener funciones personalizadas para resolver un problema específico de tu aplicación. Ambos tipos de módulos son útiles, pero su alcance y propósito son diferentes: los estándar son herramientas generalizadas, mientras que los módulos propios están orientados a necesidades puntuales.

Por otro lado, los módulos estándar también se distinguen de los módulos creados por terceros, como **colorama**. Estos últimos son desarrollados por la comunidad o empresas externas y suelen resolver problemas más específicos que los módulos estándar no abordan. A diferencia de los módulos estándar, los de terceros necesitan instalarse por separado mediante herramientas como **pip**. Sin embargo, al igual que los módulos estándar, amplían las capacidades de Python y se integran perfectamente con el lenguaje. Por ejemplo, mientras el módulo estándar **math** ofrece operaciones matemáticas avanzadas, un módulo de terceros como **numpy** lleva el cálculo numérico a otro nivel.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Característica	Módulos Estándar	Módulos Propios	Módulos de Terceros
Disponibilidad	Incluidos en la instalación de Python, no requieren configuración adicional.	Creación específica para un proyecto.	Necesitan instalarse con herramientas como **pip**.
Aplicabilidad	General y aplicable a diversos contextos.	Limitada a las necesidades del proyecto donde se crean.	Resolución de problemas específicos o avanzados no cubiertos por los módulos estándar.
Ejemplo	**random**, **datetime**, **math**.	Un archivo **utilidades.py** con funciones personalizadas.	**colorama** para manejar colores en la consola, **requests** para solicitudes HTTP.
Mantenimiento	Gestionado por los desarrolladores de Python, con documentación oficial.	Totalmente bajo la responsabilidad del programador.	Mantenidos por la comunidad o empresas, con documentación variable.

Los módulos estándar son herramientas confiables, versátiles y siempre disponibles. Son diferentes a los módulos creados por el usuario, que tienen un alcance limitado a las necesidades del proyecto, y a los de terceros, que aportan funcionalidades adicionales a través de instalación externa.

Comprender estas diferencias te permitirá aprovechar cada tipo de módulo según el contexto y las necesidades de tu proyecto.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción al módulo random.

El **módulo random** es una herramienta estándar en Python que te permite trabajar con números aleatorios de manera sencilla y eficaz. Este módulo es particularmente útil para tareas como simulaciones, juegos, generación de datos de prueba, y mucho más. Con **random**, podés generar números aleatorios enteros o flotantes, seleccionar elementos al azar de una lista, barajar secuencias, entre otras funcionalidades.

Ejemplo: Juego de dados
A continuación, Luis te proporciona un ejemplo que utiliza el módulo random para simular el lanzamiento de dos dados. Esto podría ser parte de un juego donde los participantes suman los valores obtenidos para determinar el resultado:

```
# Importamos el módulo random
import random

# Simulamos el lanzamiento de dos dados
def lanzar_dados():
    dado1 = random.randint(1, 6)  # Genera un número entre 1 y 6
    dado2 = random.randint(1, 6)
    suma = dado1 + dado2

    print(f"Lanzaste un {dado1} y un {dado2}. La suma es {suma}.")

# Ejecutamos el juego
lanzar_dados()
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En este ejemplo, **random.randint(1, 6)** genera un número entero aleatorio entre 1 y 6, simulando el resultado de un dado. La suma de los valores muestra el resultado del lanzamiento.

Tabla de funciones comunes del módulo random

Función	Descripción	Ejemplo de Uso
random.random()	Genera un número flotante aleatorio entre 0.0 y 1.0.	`valor = random.random()`
random.randint(a, b)	Devuelve un número entero aleatorio entre a y b (ambos inclusive).	`dado = random.randint(1, 6)`
random.choice(seq)	Selecciona un elemento al azar de una secuencia (lista, tupla, etc.).	`elemento = random.choice([1, 2, 3, 4, 5])`
random.shuffle(seq)	Mezcla los elementos de una lista en su lugar.	`random.shuffle(lista)`
random.uniform(a, b)	Genera un número flotante aleatorio entre a y b.	`valor = random.uniform(5.5, 10.5)`
random.sample(pop, k)	Devuelve una lista de k elementos seleccionados al azar de una población.	`muestra = random.sample(range(100), 5)`

El módulo **random** no solo aporta un alto grado de flexibilidad en la generación de datos aleatorios, sino que también facilita la creación de programas interactivos y cambiantes. Al dominar sus funciones más comunes, podés agregar elementos de sorpresa y variabilidad a tus aplicaciones, haciendo que sean más interesantes.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción al módulo math

El módulo **math** en Python proporciona un conjunto de funciones matemáticas avanzadas y constantes que son útiles para realizar cálculos científicos y matemáticos. Este módulo también es parte de los módulos estándar de Python, lo que significa que no necesitas instalarlo por separado.

Un ejemplo típico de uso del módulo math es calcular la raíz cuadrada de un número, encontrar el valor del seno o coseno de un ángulo, o trabajar con constantes matemáticas como π (pi) o e (la base del logaritmo natural).

Ejemplo práctico: Calcular el área de un círculo

Imaginá que querés calcular el área de un círculo dado su radio. Usando el módulo **math**, podés hacer lo siguiente:

```
# Importamos el módulo math
import math

# Pedimos al usuario que ingrese el radio del círculo
radio = float(input("Ingresá el radio del círculo: "))

# Calculamos el área usando la fórmula: área = π * radio^2
area = math.pi * math.pow(radio, 2)

# Mostramos el resultado
print(f"El área del círculo con radio {radio} es: {area:.2f}")
```

Este código comienza importando el módulo **math**, lo que permite acceder a sus funciones y constantes, como el valor de π. Luego, se solicita el ingreso del radio del círculo para proceder con el cálculo del área. Para ello, se utiliza la fórmula matemática del área del círculo: π multiplicado por el radio elevado al cuadrado. En el código, esto se implementa

utilizando **math.pi** para obtener el valor de π y **math.pow()** para calcular el cuadrado del radio. Finalmente, se muestra el área calculada en pantalla, formateada con dos decimales para una presentación más clara.

Tabla de funciones comunes del módulo math

Función	Uso	Ejemplo
math.sqrt(x)	Calcula la raíz cuadrada de x,	`math.sqrt(16) devuelve 4.0.`
math.pow(x, y)	Eleva x a la potencia y.	`math.pow(2, 3) devuelve 8.0.`
math.pi	Devuelve el valor de π (constante matemática).	`math.pi devuelve 3.141592653589793.`
math.e	Devuelve el valor de e (base de logaritmos naturales).	`math.e devuelve 2.718281828459045.`
math.sin(x)	Calcula el seno de x (en radianes).	`math.sin(math.pi / 2) devuelve 1.0.`
math.cos(x)	Calcula el coseno de x (en radianes)	`math.cos(0) devuelve 1.0.`
math.log(x, base)	Calcula el logaritmo de x en la base especificada (por defecto, e).	`math.log(100, 10) devuelve 2.0.`
math.ceil(x)	Redondea x hacia arriba al entero más cercano.	`math.ceil(3.2) devuelve 4.`
math.floor(x)	Redondea x hacia abajo al entero más cercano.	`math.floor(3.8) devuelve 3.`

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Con el módulo math, podés realizar cálculos precisos y rápidos, facilitando tareas que serían más complejas sin estas herramientas.

Introducción al módulo datetime

El módulo **datetime** de Python es una herramienta esencial para trabajar con fechas y horas de manera sencilla y eficiente. Proporciona una amplia gama de funciones para manipular, formatear y realizar cálculos con fechas y tiempos. A diferencia de módulos creados por el usuario o de terceros, datetime viene integrado con Python, lo que significa que no requiere instalación adicional.

Ejemplo práctico: Obtener la fecha y hora actual y formatearlas

El siguiente código muestra cómo se puede usar datetime para obtener la fecha y hora actual y presentarlas en diferentes formatos:

```
import datetime

# Obtener la fecha y hora actual
fecha_hora_actual = datetime.datetime.now()

# Formatear y mostrar la fecha y hora
print("Fecha y hora actual:", fecha_hora_actual)
print("Solo la fecha:", fecha_hora_actual.strftime("%Y-%m-%d"))
print("Solo la hora:", fecha_hora_actual.strftime("%H:%M:%S"))

# Mostrar la fecha en formato legible
print("Fecha legible:", fecha_hora_actual.strftime("%d de %B de %Y"))
```

Este código comienza importando el módulo **datetime**. Luego, utiliza la función **datetime.datetime.now()** para obtener un objeto que representa la fecha y hora actual. Este objeto se puede formatear utilizando el método **.strftime()**, que permite personalizar la salida según un formato especificado. Por ejemplo, "**%Y-%m-%d**" produce una salida en formato de fecha estándar, mientras que "**%H:%M:%S**" muestra únicamente la hora. Además, se utiliza "**%d de %B de %Y**" para generar una fecha en un formato más legible, como "26 de diciembre de 2024".

Tabla de funciones comunes del módulo datetime

Función	Descripción	Ejemplo
datetime.now()	Obtiene la fecha y hora actual del sistema.	`datetime.datetime.now()`
datetime.date.today()	Obtiene solo la fecha actual.	`datetime.date.today()`
strftime(formato)	Convierte un objeto **datetime** en un string con el formato dado.	`.strftime("%d/%m/%Y")`
timedelta	Permite realizar operaciones aritméticas con fechas y tiempos.	`datetime.timedelta(days=5)`
datetime.strptime(cadena, formato)	Convierte un string a un objeto **datetime** siguiendo un formato dado.	`datetime.datetime.strptime("2024-12-26", "%Y-%m-%d")`

El módulo datetime es parte de la biblioteca estándar de Python, lo que significa que no es necesario instalarlo por separado. Este módulo está incluido de manera predeterminada en cualquier instalación de Python, listo para ser utilizado.

El módulo **datetime** es una herramienta fundamental para cualquier proyecto que necesite trabajar con tiempo, como sistemas de registro de eventos, aplicaciones de calendario, o análisis de datos temporales.

Introducción al módulo colorama

El módulo **colorama** es una librería de terceros que permite agregar colores y estilos al texto mostrado en la consola. Esto es especialmente útil cuando querés destacar información importante o hacer que tu programa sea más atractivo visualmente. A diferencia de los módulos estándar como **datetime**, colorama no viene incluido por defecto en Python, por lo que es necesario instalarlo antes de poder usarlo.

¿Por qué usar colorama?

En muchas aplicaciones, los mensajes en consola son fundamentales para la interacción con el quién usa el programa o la depuración del código. Sin embargo, el texto plano puede ser difícil de seguir o interpretar rápidamente. **colorama** permite agregar colores al texto, haciéndolo más claro y visualmente organizado. Por ejemplo, podés resaltar mensajes de error en rojo, advertencias en amarillo y mensajes de éxito en verde.

Instalación de colorama

Para instalar colorama, necesitás usar el gestor de paquetes de Python, **pip**. Asegurate de tener una conexión a internet y seguí estos pasos:

1. Abrí tu terminal o consola.
2. Escribí el siguiente comando y presioná Enter:

```
pip install colorama
```

3. Verificá que la instalación se completó correctamente observando el mensaje que aparece en la consola, donde deberías ver algo como "*Successfully installed colorama*".

Si querés verificar que colorama está disponible en tu entorno, podés usar el siguiente comando:

```
pip show colorama
```

Esto mostrará información sobre la versión instalada y su ubicación.

Cómo usar colorama

Para empezar a darle color a tus textos en la terminal con Colorama, el primer paso es inicializar la biblioteca. Esto se logra mediante una llamada a la función **init()**. Al hacerlo, **colorama** prepara tu entorno para que pueda interpretar y mostrar los colores correctamente. Veamos cómo hacerlo:

```
from colorama import init
init()
```

Una vez que **colorama** está inicializado, cambiar el color del texto es tan simple como utilizar una de las constantes de **colorama.Fore.***. Por ejemplo, si deseas imprimir el texto "Hola Mundo" en color verde, podés hacerlo de la siguiente manera:

```
from colorama import Fore, init
init()
print(Fore.GREEN + "Hola Mundo")
```

Este código agrega el color verde al texto, resultando en una salida colorida y llamativa en tu terminal.

```
Hola Mundo
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Personalización avanzada con colorama.

Cambiar el fondo del texto

Al igual que modificamos el color del texto con **colorama.Fore.*,** cambiar el color de fondo es igual de sencillo usando **colorama.Back.***. Por ejemplo, para establecer un fondo rojo, harías lo siguiente:

```
from colorama import Fore, Back, init
init()
print(Back.RED + "Hola Mundo")
```

Salida:

```
Hola Mundo
```

Combinar colores de texto y fondo

Colorama también te permite combinar colores de texto y fondo en una misma impresión:

```
from colorama import Fore, Back, init
init()
print(Fore.BLUE + Back.YELLOW + "Hola Mundo")
```

Salida:

```
Hola Mundo
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejercicio Práctico

Cómo es habitual, finalizas el día con una reunión de trabajo en la oficina de Mariana, que te felicita por tu progreso. "*Luis me mostró el programa de gestión de frutas que desarrollaste, y estoy muy impresionada con lo bien que aplicaste las funciones para organizar el código. Sin embargo, es hora de mejorarlo*:

Tu tarea es volver a modificar el programa que escribiste la clase anterior, para que realice las mismas tareas, pero que su interacción mediante la terminal esté mejorada utilizando Colorama. También sería genial si agregas un dato a cada producto que contenga la fecha y hora de la compra, usando la librería datetime.

¡Estoy segura que te va a gustar el nuevo desafío!

Materiales y Recursos Adicionales:

Artículos:
Sitio oficial Python: Módulos estándar
Datacamp: Tutorial sobre cómo trabajar con módulos en Python

Videos:
Byspel: Cómo crear un módulo propio
Cosas de Informatica: Descubre cómo crear y utilizar módulos en Python

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Preguntas para Reflexionar:

1. ¿Cómo creés que el uso de módulos puede ayudarte a mejorar la organización de tus proyectos? Pensá en ejemplos concretos donde dividir el código en módulos podría simplificar su mantenimiento.
2. Los módulos estándar como random y datetime ofrecen muchas funcionalidades predefinidas. ¿Qué ventajas encontrás al usar estas herramientas en lugar de desarrollar todo desde cero? ¿Podrías pensar en aplicaciones prácticas para estos módulos en tus proyectos?
3. La biblioteca colorama permite personalizar la interacción en la consola con colores. ¿Cómo podría mejorar la experiencia del usuario en programas que diseñes? ¿Qué otras aplicaciones creativas podrías imaginar para este módulo?
4. Ahora que entendiste cómo crear tus propios módulos, ¿cómo los usarías para estructurar un proyecto más grande? ¿Qué partes del código podrías separar en diferentes archivos para facilitar el trabajo en equipo?

Próximos Pasos:

En la próxima clase, vas a explorar un concepto fundamental en el desarrollo de software: **la persistencia de datos**. Aprenderás cómo hacer que tu programa guarde información de forma permanente, permitiendo que los datos ingresados se conserven incluso después de cerrar la aplicación. Esta habilidad es indispensable para desarrollar aplicaciones que gestionen información de manera útil y práctica.

Vamos a trabajar con archivos de texto, aprendiendo a leerlos y escribirlos directamente desde Python. También aprenderás el manejo de excepciones con **try-except**, una herramienta que te permitirá anticiparte a posibles errores y hacer tu código más robusto y confiable. Además, comenzaremos a explorar el mundo de las bases de datos y SQL, entendiendo cómo estas herramientas avanzadas amplían las posibilidades de almacenamiento y gestión de información. Este nuevo conocimiento te acercará un paso más a preparar tu Trabajo Final Integrador, dotándolo de las funcionalidades esenciales que todo sistema profesional necesita. ¡Nos vemos en la próxima clase para seguir avanzando juntos!

Buenos Aires Ciudad