«Talento Tech»

Data Analytics

con Python

Clase 03

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 3 | Introducción a las Librerías para Análisis de Datos

Temario:

- Fundamentos de Python III:
    - Funciones: Scope de variables. Tipos de parámetros.
- Introducción a NumPy y Pandas. Características. Sus estructuras de datos.
- Lectura de archivos CSV con Pandas.

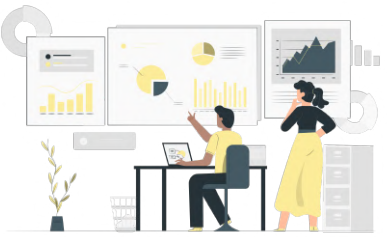
Objetivos de la clase:

- Familiarizarse con los diferentes tipos de parámetros y repasar los alcances de las variables dentro del entorno de Python.
- Conocer las estructuras de datos propias de NumPy y Pandas. Obtención de datos a partir de archivos CSV y Google Sheets.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Scope de las variables en Pyhton

El scope o alcance de las variables en Python se refiere a la **visibilidad y accesibilidad de una variable dentro de diferentes partes de un programa**. Comprender cómo funciona el scope es fundamental para evitar errores y escribir código más claro y eficiente.



Veamos un resumen sobre los **tipos de scope en Python**:

1. Tipos de Scope

a. Scope Local

Las variables que se definen dentro de una función tienen un **scope local**. Esto significa que solo son accesibles dentro de esa función. Por ejemplo:

```
def mi_funcion():
    x = 10  # Variable local
    print(x)

mi_funcion()  # Imprime 10
print(x)      # Da error, porque no definimos x fuera de la
función
```

b. Scope Global

Las variables que se definen fuera de cualquier función tienen un **scope global**. Estas variables son accesibles desde cualquier parte del código, incluyendo dentro de funciones (a menos que haya una variable local con el mismo nombre).

Por ejemplo:

```
y = 20  # Variable global
def otra_funcion():
    print(y)  # Accede a la variable global

otra_funcion()  # Imprime 20
```

Si necesitamos modificar una variable global dentro de una función, debemos usar la palabra clave global:

```
z = 30  # Variable global

def cambiar_global():
    global z  # Indica que se usa la variable global z
    z = 40

cambiar_global()
print(z)  # Imprime 40
```

c. Scope de Variables No Locales

Las variables en un **scope no local** son aquellas que se encuentran en un ámbito superior al de una función anidada, pero no son globales. Estas se acceden usando la palabra clave nonlocal:

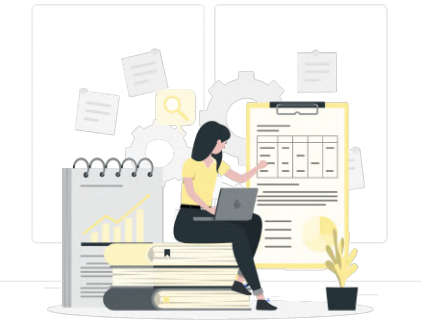
```
def funcion_externa():
    a = 5

    def funcion_interna():
        nonlocal a  # Accede a en el scope de la función externa
        a += 1
        print(a)

    funcion_interna()
```

```
funcion_externa()  # Imprime 6
```

2. Resolución del Scope

Python sigue el **principio LEGB** para la resolución del scope, que se basa en el siguiente orden:

- **Local:** Variables definidas en la función actual.
- **Enclosing (cerramiento):** Variables en el scope de funciones externas.
- **Global:** Variables definidas en el ámbito global.
- **Built-in:** Variables y funciones incorporadas en Python.



Este orden determina cómo Python busca el nombre de una variable y en qué nivel puede encontrarla.

Reflexión final

El entendimiento del **scope de las variables** es esencial para prevenir problemas como conflictos de nombres o errores de referencia. Al trabajar con diferentes scopes, asegúrate de utilizar las palabras clave adecuadas (global, nonlocal) cuando sea necesario, y organiza tu código de manera que las variables sean accesibles sólo donde se necesiten.

Tipos de parámetros en las funciones de Python

En Python, las funciones pueden recibir argumentos de diferentes formas, lo que permite mayor flexibilidad y claridad en tu código. A continuación, se explican los diferentes **tipos de parámetros que puedes utilizar en las funciones:** parámetros posicionales, parámetros por defecto, parámetros indefinidos `(*args)` y parámetros de palabras clave indefinidas `(**kwargs)`.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

1. Parámetros Posicionales

Los parámetros posicionales son los más comunes. Se definen en la declaración de la función y deben ser pasados en el mismo orden en que están definidos.

```
def suma(a, b):
    return a + b

resultado = suma(3, 5)  # a=3 y b=5
print(resultado)  # Imprime 8
```

En este ejemplo, `a` y `b` son parámetros posicionales. La función espera recibir dos argumentos, que deben ser proporcionados en el orden correcto.

2. Parámetros por Defecto

Los parámetros por defecto tienen valores predefinidos. Si no se proporciona un argumento al llamar a la función, se usa el valor por defecto.

```
python

def saludar(nombre="Mundo"):
    return f"Hola, {nombre}!"

print(saludar())          # Imprime "Hola, Mundo!"
print(saludar("Alice"))   # Imprime "Hola, Alice!"
```

Aquí, el parámetro `nombre` tiene un valor por defecto de "Mundo". Si no se proporciona un valor al llamar a `saludar`, se usará este valor.

3. Parámetros Indefinidos (*args)

Los parámetros indefinidos se utilizan cuando no sabes el número exacto de argumentos que se van a pasar a la función. Con `*args`, puedes captar un número variable de argumentos posicionales, que se recopilan en una tupla.

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

print(sumar_todos(1, 2, 3))           # Imprime 6
print(sumar_todos(4, 5, 6, 7, 8))      # Imprime 30
```

En este ejemplo, `*args` permite que la función `sumar_todos` acepte cualquier cantidad de argumentos, los cuales se suman y devuelven.

4. Parámetros de Palabras Clave Indefinidas (**kwargs)

De manera similar a `*args`, `**kwargs` permite capturar un número variable de argumentos, pero en este caso, esos argumentos son tratados como pares de clave-valor. Recopilados en un diccionario.

```
def mostrar_informacion(nombre, **kwargs):
    info = f"Nombre: {nombre}\n"
    for clave, valor in kwargs.items():
        info += f"{clave}: {valor}\n"
    return info

print(mostrar_informacion("Alice",     edad=30,     ciudad="Madrid",
profession="Ingeniera"))
```

En este ejemplo, `**kwargs` permite que la función `mostrar_informacion` reciba cualquier cantidad de argumentos adicionales. La salida mostrará el nombre junto con la información adicional pasada.

5. Combinando Parámetros

Puedes combinar todos estos tipos de parámetros en una sola función, pero es importante seguir el orden correcto al definir los parámetros: primero los posicionales, luego los parámetros por defecto, seguidos de `*args`, `**kwargs`. Aquí hay un ejemplo que ilustra esto:

```
def funcion_compleja(param1, param2=10, *args, kwarg1=None,
**kwargs):
    print(f"param1: {param1}, param2: {param2}, args: {args},
kwarg1: {kwarg1}, kwargs: {kwargs}")

funcion_compleja(5, 20, 30, 40, kwarg1='Valor1',
otro_kwarg='Valor2')

'''
Salida esperada
yaml

param1: 5, param2: 20, args: (30, 40), kwarg1: Valor1, kwargs:
{'otro_kwarg': 'Valor2'}
'''
```

Resumen

1. **Parámetros Posicionales:** Reciben valores en un orden específico.
2. **Parámetros por Defecto:** Tienen un valor asignado que se utiliza si no se proporciona argumento.
3. **Parámetros Indefinidos `*args`:** Permiten un número variable de argumentos posicionales.
4. **Parámetros de Palabras Clave Indefinidas `**kwargs`:** Permiten un número variable de argumentos como pares clave-valor.

Comprender estos tipos de parámetros contribuye a generar funciones más flexibles y legibles.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

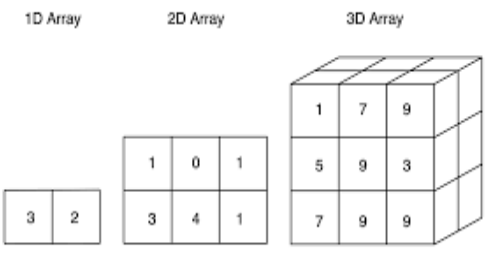
Introducción a NumPy y Pandas

En el campo del análisis de datos, Python se consolidó como uno de los lenguajes más populares, y dos bibliotecas destacan por su importancia: **NumPy** y **Pandas**. Ambas ofrecen potentes herramientas que **facilitan la manipulación y el análisis de datos**.

NumPy

NumPy, que significa **"Numerical Python"**, se centra principalmente en el **manejo de datos numéricos**. Es una biblioteca fundamental que ofrece soporte para arrays multidimensionales y matrices, junto con una amplia colección de funciones matemáticas para operar sobre estos objetos. Una de las características más destacadas de NumPy es su **capacidad para realizar cálculos vectorizados**, lo que permite ejecutar operaciones en datos completos a la vez, en vez de hacerlo elemento por elemento, lo que resulta en un rendimiento considerablemente mejorado.

La estructura de datos más importante en NumPy es el **array**. A diferencia de las listas de Python, los arrays de NumPy son más eficientes en términos de memoria y están diseñados para realizar cálculos matemáticos muy rápidamente. Existen varios tipos de arrays, como los arrays unidimensionales (vectores) y bidimensionales (matrices), pero también podemos crear arrays de más dimensiones. Esto hace que NumPy sea una **herramienta excelente para trabajar con grandes volúmenes de datos**.



Buenos Aires aprende
Agencia de Habilidades para el Futuro

Pandas

Pandas, cuyo nombre proviene de **"Panel Data"**, se construyó sobre NumPy y está diseñado para el **manejo y análisis de datos estructurados**. Aporta características que simplifican el trabajo con datos en forma de tablas, similares a las que se encuentran en una hoja de cálculo o en bases de datos.

La principal estructura de datos en Pandas es el **DataFrame**. Un DataFrame es una colección de datos organizados en filas y columnas, donde cada columna puede contener un tipo de dato diferente (números, cadenas, fechas, etc.). Pandas también tiene otra estructura llamada **Series**, que es esencialmente una columna única de un DataFrame. Esto proporciona una manera intuitiva de trabajar con datos, ya que puedes acceder y manipular tanto los valores como las etiquetas, lo que facilita el análisis exploratorio y la limpieza de datos.

```
In [10]: ventas = pd.DataFrame({
             "Entradas": [41, 32, 56, 18],
             "Salidas": [17, 54, 6, 78],
             "Valoración": [66, 54, 49, 66],
             "Límite": ["No", "Sí", "No", "No"],
             "Cambio": [1.43, 1.16, -0.67, 0.77]
         },
             index = ["Ene", "Feb", "Mar", "Abr"]
         )
         ventas
```

Out[10]:

	Entradas	Salidas	Valoración	Límite	Cambio
Ene	41	17	66	No	1.43
Feb	32	54	54	Sí	1.16
Mar	56	6	49	No	-0.67
Abr	18	78	66	No	0.77

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Características Clave

NumPy

- **Eficiencia**: Los arrays de NumPy son más eficientes en términos de velocidad y espacio en memoria en comparación con las listas de Python.
- **Funciones Matemáticas**: Proporciona una amplia gama de funciones matemáticas y estadísticas.
- **Vectorización**: Permite operaciones aritméticas rápidas y eficientes en arrays completos.

Pandas

- **Facilidad de Uso**: Proporciona estructuras de datos fáciles de usar y microestructuras para manipulación y análisis.
- **Manipulación de Datos**: Ofrece herramientas para filtrar, agrupar, y agrupar datos eficientemente.
- **Integración**: Puede integrarse fácilmente con otras bibliotecas de Python y es compatible con datos de diferentes formatos, como CSV, Excel y bases de datos SQL.

Reflexión Final

Para un analista de datos en Python, **NumPy** y **Pandas** son herramientas indispensables. Mientras que NumPy se enfoca en el manejo eficiente de datos numéricos a través de arrays, Pandas ofrece estructuras de datos más complejas y flexibles como DataFrames, que permiten trabajar con datos de manera más sencilla. Juntas, **estas bibliotecas forman la base sobre la que se construyen muchos proyectos de análisis de datos, procesamiento de datos y aprendizaje automático**.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Materiales y recursos adicionales

- NumPy Documentation
- Pandas Documentation

Próximos pasos

- Concepto de calidad de los datos y su importancia.
- Identificación y tratamiento de valores nulos y duplicados.

Ejercicios prácticos

SYNTHDATA
SINTETIZA CONOCIMIENTO

Actividad 1: Tipo de Parámetros

Contexto
El mentor a cargo de esta actividad es Matías, Data Analyst en SynthData. Su enfoque estará en ayudarte a comprender cómo funcionan los parámetros en la programación, ya que son esenciales para manipular datos de manera efectiva.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Objetivos

- Aprender a crear funciones con diferentes tipos de parámetros en Python.
- Practicar la implementación de funciones para resolver problemas comunes de datos.

Ejercicio práctico

1. **Parámetros Posicionales**: Crear una función llamada `multiplicar` que reciba dos números y devuelva su producto. Probar la función con diferentes pares de números.
2. **Parámetros por Defecto:** Crear una función llamada `bienvenida` que reciba un nombre y un mensaje de bienvenida. El mensaje debe tener un valor por defecto de "¡Bienvenido!". Si no se proporciona un mensaje, la función debe usar el valor por defecto. Probar la función proporcionando solo el nombre y con el mensaje.
3. **Parámetros Indefinidos `(*args)`:** Crear una función llamada `calcular_promedio` que acepte un número variable de notas (posicionales) y devuelva el promedio de esas notas. Probar la función con diferentes cantidades de notas.
4. **Parámetros de Palabras Clave Indefinidas `(**kwargs)`:** Crear una función llamada `concatena_info` que reciba un nombre y un número indefinido de palabras clave (por ejemplo, edad, ciudad, ocupación) y devuelva un string que combine todos los datos. Probar la función pasando varios pares de clave-valor.

¿Por qué importa esto en SynthData?

Entender los diferentes tipos de parámetros en Python permite escribir funciones más flexibles y reutilizables. En el contexto de SynthData, muchas de las soluciones de análisis requieren que los datos se procesen de diferentes maneras según las necesidades de los consumidores de los datos. Al dominar estos conceptos, podrás colaborar mejor con tu equipo, facilitando la integración y análisis de datos.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Actividad 2: Introducción a NumPy y Pandas

Contexto
La mentoría para esta actividad estará a cargo de Luis, el Analista de BI. Luis te guiará en la carga y manipulación de datos utilizando Pandas, una herramienta para el análisis de datos.

Objetivos

- Aprender a cargar y visualizar archivos CSV, Google Sheets y Excel utilizando Pandas.
- Familiarizarse con las operaciones básicas de visualización de datos.

Ejercicio práctico

1. Cargar un csv en Google Colab y visualizar sus primeras filas usando Pandas.
2. Repetir el ejercicio anterior con una planilla de Google Sheets.
3. Investigá el método para cargar planillas de Excel en Google Colab, utilizando Pandas, y repetí el ejercicio del punto 1.

Sets de datos

1. ventas.csv
2. ventas
3. Utiliza una planilla de excel que tengas en tu PC

¿Por qué importa esto en SynthData?
Aprender a cargar datos es uno de los primeros pasos para el análisis de datos. Aprender a usar Pandas te permitirá manipular y analizar grandes conjuntos de datos de manera eficiente.

⛔ **Estos ejercicios son una simulación de cómo se podría resolver el problema en este contexto específico. Las soluciones encontradas no aplican de ninguna manera a todos los casos.**
Recordá que las soluciones dependen de los sets de datos, el contexto y los requerimientos específicos de los stakeholders y las organizaciones.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad