«Talento Tech»

Data Analytics

con Python

Clase 02

Buenos Aires Ciudad

Clase N° 2 | Estructuras de Datos en Python

Temario:

- Listas, tuplas, diccionarios y conjuntos.
- Uso de estas estructuras para almacenamiento y manipulación de datos.
- Fundamentos de Python II:
    a. funciones flecha
    b. list y dict comprehension

Objetivos de la clase:

- Conocer los tipos de datos compuestos en Python.
- Aprender a crear estructuras de datos con funciones compactas y eficientes.

Estructuras de datos en Python

Python ofrece varias estructuras de datos integradas que permiten almacenar y manipular colecciones de datos de manera eficiente. Las estructuras más comunes son las listas, tuplas, diccionarios y conjuntos. Cada una tiene sus características particulares que las hacen adecuadas para diferentes tipos de tareas.

Listas

Las listas son **colecciones ordenadas y mutables**, lo que significa que podemos modificar su contenido después de haberlas creado. Se definen utilizando corchetes `[]`. Las listas pueden contener elementos de diferentes tipos, incluyendo otros objetos, lo que las convierte en una herramienta muy versátil.

```
mi_lista = [1, 2, 3, "cuatro", 5.0]
mi_lista.append(6)  # Agrega un nuevo elemento
print(mi_lista)  # Imprime: [1, 2, 3, 'cuatro', 5.0, 6]
```

Las listas son útiles cuando necesitamos mantener el orden de los elementos o realizar operaciones como agregar, eliminar o modificar elementos.

Tuplas

Las tuplas son similares a las listas, pero son **inmutables**: una vez que se crea una tupla, no se puede modificar. Se definen utilizando paréntesis `()`. Dicha característica las hace ideales para **almacenar datos que no deben cambiar**, como coordenadas o configuraciones constantes.

Ejemplo:

```
mi_tupla = (1, 2, 3, "cuatro")
# mi_tupla[0] = 10  # Esto causaría un error, ya que no se
puede modificar
print(mi_tupla)  # Imprime: (1, 2, 3, 'cuatro')
```

A menudo, las tuplas se utilizan cuando necesitas asegurar que los datos originales permanezcan sin cambios.

Diccionarios

Los diccionarios son **estructuras de datos que almacenan pares de clave-valor**. Se definen utilizando llaves {} y son **mutables**. Esto significa que puedes agregar, modificar y eliminar elementos a tu conveniencia. Los diccionarios son ideales para almacenar datos que necesitan ser accedidos de manera rápida y eficiente mediante una clave única.

Ejemplo:

```
mi_diccionario = {"nombre": "Juan", "edad": 30, "ciudad":
"Madrid"}
mi_diccionario["edad"] = 31  # Modifica el valor asociado a
la clave 'edad'
mi_diccionario["profesión"] = "Ingeniero"  # Agrega una nueva
clave-valor
print(mi_diccionario)  # Imprime: {'nombre': 'Juan', 'edad':
31, 'ciudad': 'Madrid', 'profesión': 'Ingeniero'}
```

Los diccionarios son especialmente útiles cuando necesitas organizar datos en un formato más estructurado que una lista.

Conjuntos

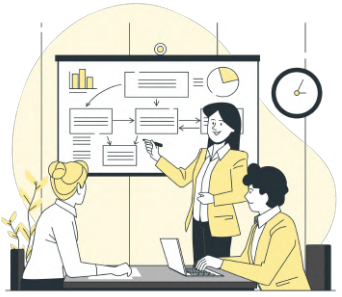
Los conjuntos son **colecciones no ordenadas de elementos únicos**, lo que significa que no pueden contener duplicados. Se definen usando llaves `{}` o la función `set()`. Los conjuntos son valiosos para realizar operaciones matemáticas como uniones, intersecciones y diferencias.

Ejemplo:

```
mi_conjunto = {1, 2, 3, 3, 4}
print(mi_conjunto)  # Imprime: {1, 2, 3, 4} (el valor 3 solo
aparece una vez)
mi_conjunto.add(5)  # Agrega un nuevo elemento
print(mi_conjunto)  # Imprime: {1, 2, 3, 4, 5}
```

Los conjuntos son útiles en situaciones donde el orden no es importante, pero se necesita garantizar la unicidad.

Fundamentos de Python



Para sacarle el máximo rendimiento a estas estructuras de datos, es crucial entender algunos conceptos fundamentales de Python, como las **funciones lambda** y las **comprensiones de listas y diccionarios**.

Funciones Lambda

Las funciones lambda son una **forma concisa de definir funciones pequeñas y anónimas en Python**. Se utilizan comúnmente para operaciones rápidas y se definen con la palabra clave `lambda`. Estas funciones son particularmente útiles en combinación con funciones de orden superior como `map()` y `filter()`.

Ejemplo:

```
suma = lambda x, y: x + y
print(suma(5, 3))  # Imprime: 8
```

Las funciones lambda permiten crear funciones en línea sin necesidad de definirlas completamente con `def`.

List Comprehension

Las list comprehensions ofrecen una **forma elegante y concisa de crear listas a partir de otras listas**. Utilizan una sintaxis compacta que permite aplicar una operación a cada elemento de una colección y crear una nueva lista en una sola línea.

Ejemplo:

```
numeros = [1, 2, 3, 4, 5]
cuadrados = [x**2 for x in numeros]
print(cuadrados)  # Imprime: [1, 4, 9, 16, 25]
```

Esta característica es especialmente útil para transformar y filtrar datos de una manera rápida y legible.

Dict Comprehension

De manera similar a las comprensiones de listas, las comprensiones de diccionarios te **permiten construir diccionarios de manera compacta**. Esto es fundamental cuando quieres transformar o filtrar datos basados en condiciones específicas.

Ejemplo:

```
numeros = [1, 2, 3, 4, 5]
cuadrados_dict = {x: x**2 for x in numeros}  # Mtira como clave el número y como valor su cuadrado
print(cuadrados_dict)  # Imprime: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25
```

Utilizar diccionarios de esta forma es muy eficiente, especialmente en aplicaciones de análisis de datos.

Reflexión Final

Dominar estas estructuras de datos y conceptos fundamentales en Python es esencial para cualquier aspirante a analista de datos. Permiten un **almacenamiento organizado** y una **manipulación de datos efectiva**, lo que es crucial para el análisis de grandes conjuntos de datos. Al familiarizarte con listas, tuplas, diccionarios y conjuntos, así como con las funciones lambda y las comprensiones, estarás mejor preparado para abordar problemas complejos en tus proyectos de Data Analytics.

Material de referencia

- Python Data Types (W3 Schools)

Próximos Pasos

- 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.

Ejercicios Prácticos

Actividad: Manipulación de Estructuras de Datos en Python

Contexto

¡Bienvenido a esta emocionante actividad en tu proceso de selección en SynthData! Esta vez, trabajarás en la manipulación de diferentes estructuras de datos en Python, contando con la guía de Sabrina, nuestra Data Engineer. Este ejercicio es ideal para fortalecer tus habilidades en el manejo y transformación de datos con Python, lo cual es esencial para tus futuras tareas en el análisis de datos.
En esta actividad, te centrarás en crear y manipular diferentes estructuras de datos en Python. Desde listas y diccionarios, hasta el uso de funciones y comprehensions, tendrás la oportunidad de experimentar de manera práctica con estas herramientas. Al finalizar, estarás más preparado para aplicar estas técnicas en proyectos del mundo real.

Objetivos

- Crear y manipular diversas estructuras de datos, aplicando tanto funciones normales como funciones lambda.
- Utilizar comprehensions para crear listas y diccionarios, comparando estos enfoques con los métodos tradicionales.

Ejercicio Práctico

1. **Estructuras de Datos**
    a. Crear una lista llamada `numeros` que contenga los números del 1 al 10.

b. Crear una tupla llamada `meses` que contenga los nombres de los 12 meses del año.
c. Crear un diccionario llamado `notas` con el nombre de tres estudiantes como claves y sus respectivas notas como valores.
d. Crear un conjunto llamado `numeros_unicos` con algunos números, asegurándote de que no haya duplicados.

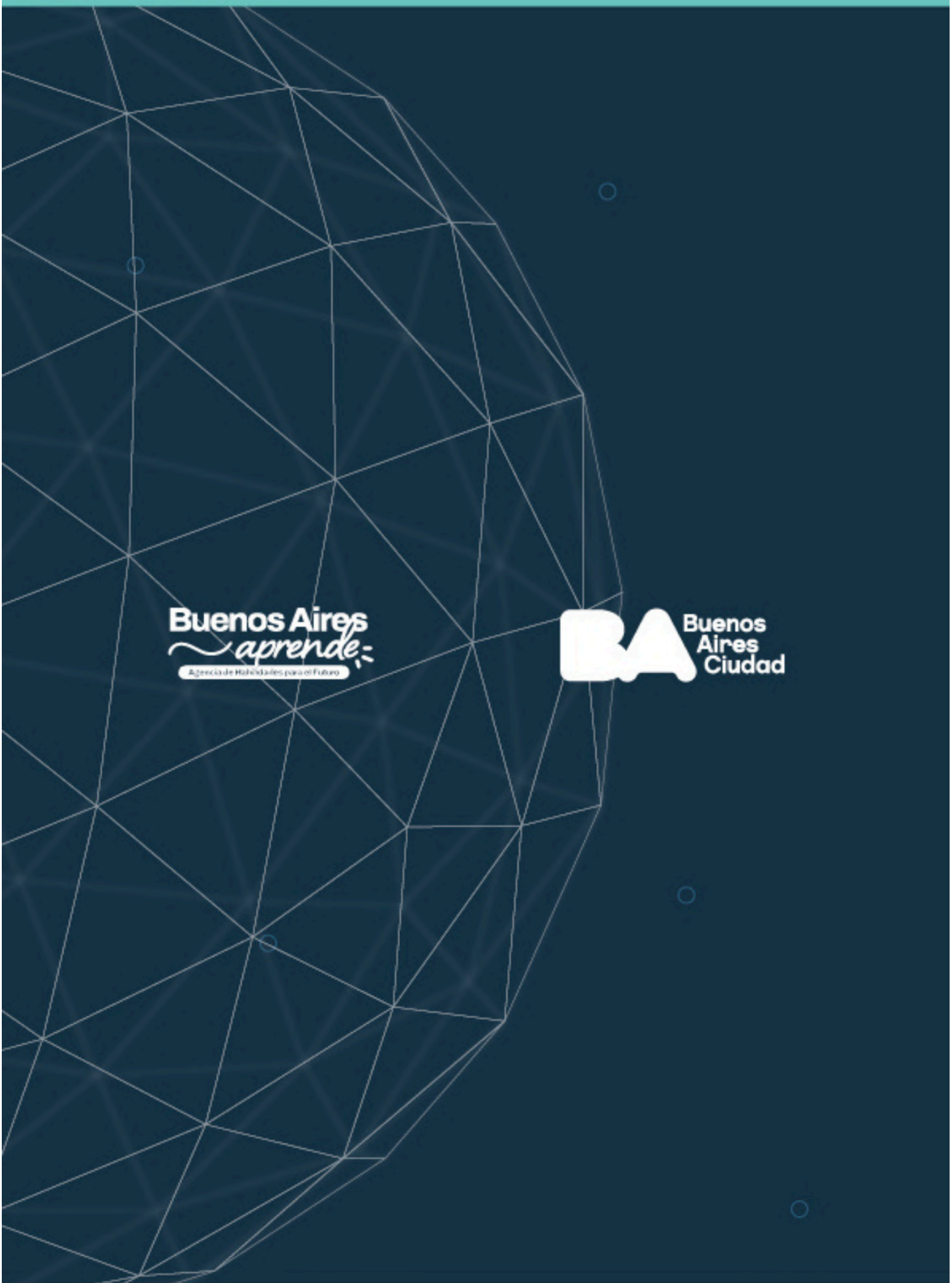
2. **Modificación y Acceso**
   a. Modificar uno de los valores del diccionario `notas` (cambiar la nota de uno de los estudiantes).
   b. Acceder a los elementos de la lista `numeros` y mostrar todos los números pares utilizando un bucle.
   c. Acceder al primer mes de la tupla `meses` y al último mes. Mostrar ambos.

3. **Funciones Normales y Funciones Lambda**
   a. Definir una función normal llamada `multiplicar_por_dos` que tome un número y lo multiplique por 2. Utilizar esta función para crear una nueva lista llamada `dobles`, que contenga los dobles de los números en la lista `numeros`.
   b. Definir una función lambda que haga lo mismo y utilizarla para crear una nueva lista llamada `dobles_lambda`, que contenga los dobles de los números en la lista `numeros`.
4. **Comprehensions**
   a. Utilizar list comprehensions para crear una lista llamada `cuadrados` que contenga los cuadrados de los números en la lista `numeros`.
   b. Utilizar dict comprehensions para crear un diccionario llamado `cubos` que contenga los números de la lista `numeros` como claves y sus cubos como valores.
5. **Comparación de resultados**
   a. Mostrar las listas `dobles`, `dobles_lambda` y `cuadrados`. Comparar sus resultados y argumentar cuál es el mejor método.
   b. Imprimir el diccionario `cubos` y comparar con la creación de un diccionario usando un método convencional.

¿Por qué importa esto en SynthData?

La manipulación de estructuras de datos es una habilidad fundamental para cualquier analista o ingeniero de datos. En SynthData, aprender a trabajar con listas, diccionarios, y otros tipos de datos te permitirá realizar análisis más complejos y efectivos. Sabrina te mostrará cómo aplicar diversas técnicas, como funciones lambda y comprehensions, para mejorar la eficiencia y la legibilidad de tu código. Todo esto es crucial para tener éxito en proyectos de análisis de datos.

⛔ **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