Examen Unidad I

Alumno: Ricardo Mezo Bustamante

In [ ]:
```
agenda = {}

def modificarAgenda():
    nombre = input("Ingrese el nombre: ")
    if nombre in agenda:
        print("El número de teléfono de {} es {}".format(nombre, agenda[nombre]))
        opcion = input("¿Desea modificar el número de teléfono? (S/N): ")
        if opcion.upper() == "S":
            telefono = input("Ingrese el nuevo número de teléfono: ")
            agenda[nombre] = telefono
            print("El número de teléfono de {} ha sido modificado a {}".format(nomb
    else:
        telefono = input("Ingrese el número de teléfono: ")
        agenda[nombre] = telefono
        print("El número de teléfono de {} ha sido añadido a la agenda".format(nomb


def buscarContactos():
    cadena = input("Ingrese el nombre del contacto: ")
    encontrados = [nombre for nombre in agenda.keys() if nombre.startswith(cadena)]
    print("Se han encontrado los siguientes nombres:")
    for nombre in encontrados:
        print("- {}".format(nombre))

def borrarContactos():
    nombre = input("Ingrese el nombre del contacto que desea borrar: ")
    if nombre in agenda:
        confirmar = input("¿Está seguro que desea borrar el contacto? (s/n): ")
        if confirmar == "s":
            del agenda[nombre]
            print("El contacto de {} ha sido eliminado de la agenda".format(nombre)
        else:
            print("No se ha borrado el contacto.")
    else:
        print("El contacto no existe.")

def listarContactos():
    if len(agenda) == 0:
        print("La agenda está vacía.")
    else:
        print("Contactos en la agenda:")
        for nombre, telefono in agenda.items():
            print("- {} - {}".format(nombre, telefono))

while True:
    print("-------- MENÚ --------")
    print("1.- Añadir/modificar contacto\n2.- Buscar contacto\n3.- Borrar contacto\
    opcion = input("Seleccione una opción: ")
```

```
    if opcion == "1":
        modificarAgenda()

    elif opcion == "2":
        buscarContactos()

    elif opcion == "3":
        borrarContactos()

    elif opcion == "4":
        listarContactos()

    elif opcion == "5":
        break

    else:
        print("Opción no existe...")
```

```
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Jose ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Andres ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Maria ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Carlos ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Manuel ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
El número de teléfono de Alejandro ha sido añadido a la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
Se han encontrado los siguientes nombres:
- Manuel
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
```

```
El contacto de Maria ha sido eliminado de la agenda
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
Contactos en la agenda:
- Jose - 2345678
- Andres - 456789
- Carlos - 123456789
- Manuel - 4567897
- Alejandro - 12345678
-------- MENÚ --------
1.- Añadir/modificar contacto
2.- Buscar contacto
3.- Borrar contacto
4.- Listar contactos
5.- Salir
```

Curso: Tópicos de Ciencia de Datos

Tarea: Examen Unidad 1 (Python)

Valor:40

Fecha de vencimiento: Friday, 10 de March de 2023, 20:00

Ricardo Mezo Bustamante
mezo21r@gmail.com

Cambiar usuario

7 de 15

Página 1 de 1

Entrega

Enviado para calificar
Calificado
La tarea fue enviada 10 horas 6 minutos antes
El estudiante puede editar esta entrega

Examen_Unidad_I_RMB.html

Comentarios (0)

Calificación

Calificación:

Practicas

PDF	Otro formato 0 puntos	Con formato 1 puntos	
Reporte con capturas	No envió 0 puntos	Si envió 4 puntos	

Funcionamiento de código	No funciona 0 puntos	Soluciona 1 prueba 7 puntos	Soluciona 2 pruebas 14 puntos	Soluciona 3 pruebas 21 puntos	Soluciona 4 pruebas 28 puntos	Soluciona 5 pruebas 35 puntos

Calificación actual en el libro de calificaciones

40,00

Comentarios de retroalimentación

Notificar a los estudiantes

Guardar cambios

Reiniciar

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico Superior de San Andrés Tuxtla

Ingeniería Informática

Asignatura: Tópicos de Ciencias de Datos

Semestre: Octavo Semestre

Grupo: IINF810A

Docente: MTI. Rogelio Enrique Telona Torres

Alumno: Ricardo Mezo Bustamante

-Presenta-

Investigación Unidad I

Período Escolar: Febrero - Junio 2023

San Andrés Tuxtla Ver, 12 de Marzo de 2023

Introducción

Python es un lenguaje de programación de alto nivel, interpretado, multiplataforma y orientado a objetos que se ha convertido en uno de los lenguajes más populares y utilizados en la actualidad. Una de las razones de su popularidad es su facilidad de aprendizaje, lo que lo hace ideal para principiantes en programación. Además, Python es muy versátil y se puede utilizar en una amplia variedad de aplicaciones, desde la ciencia de datos hasta la programación de videojuegos. En esta introducción, exploraremos los conceptos básicos de la programación con Python y cómo se puede utilizar para crear programas eficientes y efectivos.

Una de las principales ventajas de Python es su sintaxis simple y fácil de entender. A diferencia de otros lenguajes de programación, Python utiliza una sintaxis clara y legible, lo que hace que el código sea más fácil de leer y entender. Además, Python es un lenguaje de programación interpretado, lo que significa que no es necesario compilar el código antes de ejecutarlo. Esto hace que el proceso de programación sea más rápido y eficiente.

Otra característica importante de Python es su capacidad para ser orientado a objetos. La programación orientada a objetos es un paradigma de programación que se basa en la creación de objetos que contienen datos y métodos que actúan sobre esos datos. Python soporta la programación orientada a objetos de manera nativa, lo que significa que se pueden crear clases y objetos de manera sencilla y eficiente.

Python también es conocido por su capacidad para trabajar con módulos y paquetes. Los módulos y paquetes son archivos que contienen código Python y que se pueden importar en otros programas para su uso. Esto significa que los programadores pueden utilizar bibliotecas de código Python existentes para ahorrar tiempo y mejorar la eficiencia en la creación de programas.

Python también es utilizado en la ciencia de datos. La ciencia de datos es un campo que utiliza herramientas estadísticas y de aprendizaje automático para analizar y visualizar datos. Python es una herramienta popular en la ciencia de datos, ya que cuenta con una amplia variedad de bibliotecas y herramientas para el análisis de datos, como Pandas, NumPy y SciPy. Estas bibliotecas hacen que el análisis de datos sea más fácil y eficiente.

Python es un lenguaje de programación versátil y fácil de aprender que ofrece una amplia variedad de herramientas y bibliotecas para la programación, desde la creación de programas simples hasta el análisis de datos complejos. La sintaxis clara y legible de Python, la capacidad para trabajar con estructuras de datos complejas, la programación orientada a objetos y la facilidad para trabajar con módulos y paquetes hacen que Python sea una herramienta poderosa para cualquier programador.

Operaciones Con Python

Las operaciones en Python se refieren a la variedad de tareas y cálculos que se pueden realizar utilizando el lenguaje de programación Python. Estas operaciones incluyen operaciones matemáticas básicas, operaciones de comparación, operaciones de cadenas, operaciones de listas, operaciones de archivo y muchas otras tareas que se pueden realizar con Python.

Las operaciones matemáticas básicas en Python incluyen la suma, la resta, la multiplicación y la división de números. Estas operaciones se realizan utilizando los operadores matemáticos básicos (+, -, *, /). Python también tiene operadores matemáticos para realizar operaciones de módulo (%), exponenciación (**), división entera (//) y muchas otras operaciones matemáticas avanzadas.

Las operaciones de comparación se utilizan para comparar dos valores y determinar si son iguales, mayores que, menores que, mayores o iguales que, o menores o iguales que. Estas operaciones se realizan utilizando operadores de comparación, como ==, <, >, <= y >=. Las operaciones de comparación se utilizan comúnmente en estructuras de control de flujo, como declaraciones if y bucles.

Las operaciones de cadenas en Python se utilizan para manipular y procesar cadenas de caracteres. Algunas operaciones de cadena comunes incluyen la concatenación de cadenas, la búsqueda de subcadenas, la división de cadenas en subcadenas y la eliminación de espacios en blanco. Python también tiene métodos incorporados para convertir cadenas a minúsculas, mayúsculas y para capitalizar las letras.

Las operaciones de listas en Python se utilizan para manipular y procesar listas de elementos. Algunas operaciones de lista comunes incluyen la adición de elementos a una lista, la eliminación de elementos de una lista, la búsqueda de elementos en una lista y la ordenación de elementos en una lista. Python también tiene métodos incorporados para realizar tareas más avanzadas en las listas, como la eliminación de duplicados, la inversión de una lista y la creación de sublistas.

Las operaciones de archivo en Python se utilizan para leer y escribir archivos de texto y binarios. Algunas operaciones de archivo comunes incluyen la lectura de archivos de texto, la escritura de archivos de texto, la lectura de archivos binarios y la escritura de archivos binarios. Python también tiene funciones incorporadas para trabajar con archivos, como **open()** para abrir un archivo y **close()** para cerrar un archivo.

Python también tiene una amplia variedad de bibliotecas y módulos incorporados que permiten realizar operaciones más avanzadas en áreas específicas, como matemáticas, estadísticas, visualización de datos, aprendizaje automático y más.

Las operaciones en Python se refieren a la variedad de tareas y cálculos que se pueden realizar utilizando el lenguaje de programación Python. Desde operaciones

matemáticas básicas hasta operaciones de archivo y operaciones avanzadas en áreas específicas, Python ofrece una amplia variedad de herramientas y funciones para realizar tareas y cálculos en el mundo de la programación.

Operaciones Que No Modifican Una Lista

una lista es un tipo de dato mutable, lo que significa que se pueden realizar operaciones que modifican la lista original. Sin embargo, también hay operaciones que no modifican la lista original, sino que devuelven una nueva lista o un valor calculado. A continuación, se describen algunas de las operaciones que no modifican una lista en Python:

1. Acceder a elementos de una lista: Para acceder a un elemento específico de una lista, se utiliza la sintaxis de corchetes con el índice del elemento deseado. Por ejemplo, si **lista** es una lista, **lista[0]** devolverá el primer elemento de la lista sin modificar la lista original.

2. Slicing: El slicing se utiliza para obtener una sublista de una lista original. La sintaxis del slicing es **lista[inicio:fin]**, donde **inicio** es el índice del primer elemento de la sublista y **fin** es el índice del elemento siguiente al último de la sublista. Por ejemplo, si **lista** es una lista, **lista[0:2]** devolverá una nueva lista que contiene los dos primeros elementos de la lista original.

3. Operaciones de búsqueda: Python tiene varias funciones integradas para buscar elementos en una lista, como **index()**, **count()** y **in**. Estas funciones no modifican la lista original, sino que devuelven valores calculados o una nueva lista de resultados. Por ejemplo, **lista.index('valor')** devuelve el índice del primer elemento con el valor especificado en la lista.

4. Operaciones de ordenamiento: Python tiene varias funciones integradas para ordenar una lista, como **sorted()** y **sort()**. Estas funciones no modifican la lista original, sino que devuelven una nueva lista ordenada o ordenan la lista original. Por ejemplo, **sorted(lista)** devuelve una nueva lista ordenada, mientras que **lista.sort()** ordena la lista original.

5. Operaciones de concatenación: La concatenación de listas se utiliza para unir dos o más listas en una sola lista. La sintaxis de concatenación es **lista1 + lista2**, donde **lista1** y **lista2** son las listas a unir. La concatenación devuelve una nueva lista que contiene todos los elementos de las listas originales. Por ejemplo, **lista1 + lista2** devuelve una nueva lista que contiene todos los elementos de **lista1** y **lista2**.

6. Operaciones de copia: Python tiene varias formas de copiar una lista, como la asignación de rebanadas o el uso de la función **copy()**. Estas operaciones no modifican la lista original, sino que devuelven una nueva lista con los mismos elementos. Por ejemplo, **lista_copia = lista[:]** crea una copia de la lista original.

Operaciones Que Modifican Una Lista

Python proporciona muchas operaciones que modifican una lista existente, como añadir elementos, insertar elementos, modificar elementos, eliminar elementos, extender una lista y ordenar una lista. Estas operaciones pueden ser muy útiles en la programación, ya que permiten manipular las listas y modificarlas según sea necesario. Sin embargo, también es importante tener en cuenta que estas operaciones pueden tener un impacto en el rendimiento y en la memoria de su programa, por lo que es importante usarlas con prudencia y considerar la mejor manera de abordar el problema que se está resolviendo.

A continuación, se describen algunas de las operaciones que modifican una lista en Python:

1. Añadir elementos a una lista: Para añadir un elemento al final de una lista, se utiliza el método **append()**. Por ejemplo, si **lista** es una lista, **lista.append('nuevo_valor')** añadirá un nuevo elemento al final de la lista.
2. Insertar elementos en una lista: Para insertar un elemento en una posición específica de una lista, se utiliza el método **insert()**. Este método toma dos argumentos: el índice de la posición donde se insertará el elemento y el valor del elemento a insertar. Por ejemplo, si **lista** es una lista, **lista.insert(0, 'nuevo_valor')** insertará un nuevo elemento al inicio de la lista.
3. Modificar elementos de una lista: Para modificar un elemento específico de una lista, se utiliza la sintaxis de corchetes con el índice del elemento a modificar. Por ejemplo, si **lista** es una lista, **lista[0] = 'nuevo_valor'** modificará el primer elemento de la lista.
4. Eliminar elementos de una lista: Python tiene varios métodos para eliminar elementos de una lista, como **remove()**, **pop()** y **del**. El método **remove()** elimina el primer elemento con el valor especificado, el método **pop()** elimina el elemento en el índice especificado y devuelve su valor, y la palabra clave **del** elimina el elemento en el índice especificado. Por ejemplo, si **lista** es una lista, **lista.remove('valor_a_eliminar')** eliminará el primer elemento con el valor especificado.
5. Extender una lista con otra lista: Para añadir los elementos de otra lista al final de una lista existente, se utiliza el método **extend()**. Por ejemplo, si **lista1** y **lista2** son listas, **lista1.extend(lista2)** añadirá los elementos de **lista2** al final de **lista1**.
6. Ordenar una lista: Además de las operaciones de ordenamiento que no modifican una lista, Python tiene el método **sort()** que ordena una lista original en orden ascendente. Por ejemplo, si **lista** es una lista, **lista.sort()** ordenará la lista original en orden ascendente.

Copia De Listas En Python

La copia de listas es una técnica que se utiliza para crear una nueva lista que tenga los mismos elementos que otra lista existente, sin afectar la lista original. Esto es útil cuando se quiere modificar una lista sin cambiar la original, o cuando se quiere tener dos listas independientes con los mismos valores.

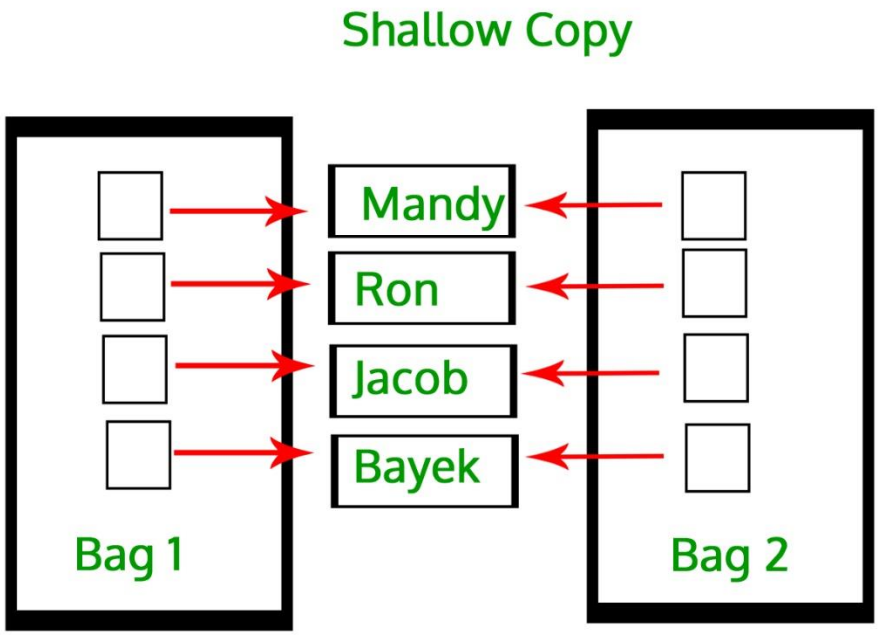
Hay dos tipos principales de copia de listas en Python: la copia superficial (shallow copy) y la copia profunda (deep copy).

1. Copia superficial (shallow copy): Una copia superficial o shallow copy en Python es una operación de clonación que crea una nueva lista, pero los elementos de la nueva lista son referencias a los mismos objetos que se encuentran en la lista original. Es decir, la nueva lista es independiente de la original, pero los elementos de ambas listas apuntan a los mismos objetos.

   Por lo tanto, cualquier cambio que se haga en un objeto referenciado en la lista original, también se reflejará en la copia superficial y viceversa. En otras palabras, las dos listas comparten los mismos objetos subyacentes.

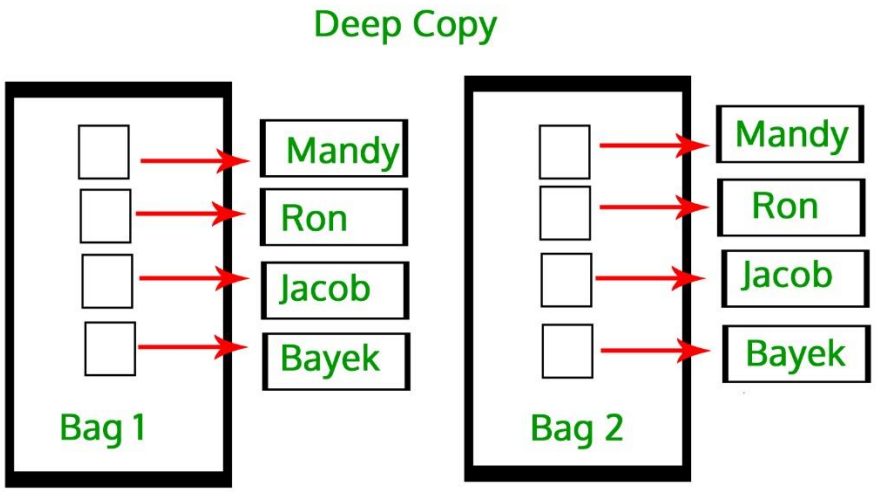
   En Python, se puede hacer una copia superficial de una lista utilizando la técnica de slicing **[:]** o el método **copy()** de la lista.

   Ahora bien, si se modifica algún elemento de la lista original, los cambios se reflejarán también en la copia superficial. Es importante tener en cuenta que si la lista original contiene objetos mutables, como listas anidadas o diccionarios, los cambios realizados en estos objetos se reflejarán en ambas listas.



2. Copia profunda (deep copy): Una copia profunda o deep copy en Python es una operación de clonación que crea una nueva lista y copia los elementos de la lista original en la nueva lista. A diferencia de una copia superficial, los elementos de la nueva lista no son referencias a los mismos objetos que se encuentran en la lista original, sino que son nuevos objetos que contienen los mismos valores.
   En otras palabras, una copia profunda crea una nueva lista independiente que no está relacionada con la lista original, y cualquier cambio que se haga en una lista no afectará a la otra. En Python, se puede hacer una copia profunda de una lista utilizando la función deepcopy() del módulo copy.

   Si se modifica algún elemento de la lista original, la copia profunda permanecerá sin cambios. Es importante tener en cuenta que hacer una copia profunda puede ser costoso en términos de recursos, especialmente para listas grandes o complejas que contengan objetos anidados. Por lo tanto, debe usarse con precaución y solo cuando sea necesario.



Operaciones Que No Modifican Un Diccionario

Los diccionarios son una estructura de datos muy útil para almacenar y manipular datos en forma de pares clave-valor. Al igual que las listas, hay operaciones que pueden llevarse a cabo en un diccionario sin modificarlo. Algunas de estas operaciones incluyen:

1. Acceder a valores: Puede acceder a los valores almacenados en un diccionario utilizando las claves correspondientes. Esto no modificará el diccionario en sí.

2. Comprobar si una clave existe: Puede comprobar si una clave está presente en un diccionario utilizando el operador in. Esto no modificará el diccionario en sí.
3. Obtener las claves o valores: Puede obtener una lista de todas las claves o valores almacenados en un diccionario utilizando los métodos **keys()** y **values()**. Esto no modificará el diccionario en sí.
4. Obtener el tamaño del diccionario: Puede obtener el número de pares clave-valor almacenados en un diccionario utilizando la función **len()**. Esto no modificará el diccionario en sí.
5. Copiar un diccionario: Puede hacer una copia de un diccionario utilizando el método **copy()**. Esto creará una copia independiente del diccionario original, que no se modificará.

Operaciones Que Modifican Un Diccionario

Es importante tener en cuenta que estas operaciones modificarán directamente el diccionario original. Por lo tanto, si desea mantener una copia del diccionario original, deberá realizar una copia antes de realizar estas operaciones. También es importante tener en cuenta que, al igual que con las listas, las operaciones de modificación de diccionarios pueden afectar la complejidad temporal y espacial del algoritmo en el que se utilizan.

Aquí hay algunas operaciones que se pueden llevar a cabo en un diccionario que lo modificarán:

1. Asignar un valor a una clave existente: Puede asignar un nuevo valor a una clave existente en un diccionario simplemente utilizando la sintaxis de asignación.

2. Agregar una nueva clave y valor: Puede agregar una nueva clave y valor a un diccionario utilizando la sintaxis de asignación.

3. Eliminar una clave y valor: Puede eliminar una clave y valor de un diccionario utilizando la palabra clave **del**.

4. Actualizar un diccionario con otro diccionario: Puede actualizar un diccionario con los valores de otro diccionario utilizando el método **update()**. Si una clave existe en ambos diccionarios, el valor en el segundo diccionario sobrescribirá el valor en el primer diccionario.

5. Eliminar todos los elementos de un diccionario: Puede eliminar todos los elementos de un diccionario utilizando el método **clear()**. Esto dejará el diccionario vacío.

Copia De Diccionarios

los diccionarios son estructuras de datos muy útiles que permiten almacenar pares clave-valor. Sin embargo, a veces es necesario copiar un diccionario para hacer cambios en él sin afectar el original. Hay dos formas de copiar un diccionario en Python: la copia superficial y la copia profunda.

La copia superficial de un diccionario crea una nueva variable que apunta a la misma memoria que el diccionario original. En otras palabras, la copia superficial copia solo la referencia al diccionario original, y no los valores reales del diccionario. Para realizar una copia superficial de un diccionario en Python, se utiliza el método **copy()**.

Por otro lado, la copia profunda de un diccionario crea una copia completa del diccionario original, incluyendo todos sus valores y claves, y todas las referencias a otros objetos. En otras palabras, la copia profunda crea un nuevo objeto que no está relacionado con el diccionario original. Para realizar una copia profunda de un diccionario en Python, se utiliza el módulo **copy** y su método **deepcopy()**.

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> #TRABAJANDO CON DICCIONARIOS (COPIA DE UN DICCIONARIO).
>>>
>>> #DICCIONARIO A COPIAR.
>>> capitales={"Italia":"Roma","Francia":"París",
              "España":"Madrid","Portugal":"Lisboa"}
>>>
>>> #CREAMOS COPIA DE DICCIONARIO("copia_capitales").
>>>
>>> copia_capitales=capitales.copy()
>>>
>>> #MOSTRAMOS DICCIONARIO ORIGINAL("capitales").
>>> print(capitales)
{'Italia': 'Roma', 'Francia': 'París', 'España': 'Madrid', 'Portugal': 'Lisboa'}
>>>
>>> #MOSTRAMOS COPIA("copia_capitales").
>>> print(copia_capitales)
{'Italia': 'Roma', 'Francia': 'París', 'España': 'Madrid', 'Portugal': 'Lisboa'}
>>>
```

Es importante tener en cuenta que la copia superficial y la copia profunda de un diccionario no son siempre necesarias. Si solo necesita hacer cambios en el diccionario sin afectar el original, puede utilizar la copia superficial. Si necesita hacer cambios en el diccionario y quiere asegurarse de que no afecten el original, puede utilizar la copia profunda.

También es importante tener en cuenta que copiar diccionarios puede tener un costo en cuanto a la complejidad temporal y espacial del algoritmo. En general, la copia profunda es más costosa que la copia superficial. Por lo tanto, es importante considerar el uso de copias superficiales o profundas según las necesidades específicas de cada caso.

Conclusión

Python es un lenguaje de programación altamente versátil y eficiente que se utiliza comúnmente en la ciencia de datos. A través de su amplia variedad de operaciones, Python permite a los usuarios manipular, procesar y analizar datos de manera eficiente. Las operaciones más comunes que se pueden realizar en Python incluyen la lectura y escritura de archivos, el procesamiento de cadenas de texto, la manipulación de estructuras de datos.

Entre las operaciones más comunes que se pueden realizar en Python se encuentran la lectura y escritura de archivos, el procesamiento de cadenas de texto, la manipulación de estructuras de datos como listas, diccionarios y tuplas, la realización de operaciones aritméticas y lógicas, y la definición de funciones y clases.

El procesamiento de cadenas de texto también es una operación importante que se puede realizar en Python. Las cadenas de texto se utilizan comúnmente en la ciencia de datos para representar etiquetas de variables, nombres de columnas y otros metadatos que se utilizan para describir los datos. Python proporciona una amplia variedad de operaciones que permiten a los usuarios manipular cadenas de texto, lo que facilita la creación y el procesamiento de metadatos.

La manipulación de estructuras de datos es otra operación importante que se puede realizar en Python. Las estructuras de datos se utilizan comúnmente en la ciencia de datos para almacenar y procesar datos. Python proporciona una amplia variedad de estructuras de datos, como listas, diccionarios y tuplas, que son fáciles de usar y altamente eficientes. Estas estructuras de datos permiten a los usuarios acceder y procesar datos de manera rápida y eficiente.

Una de las razones por las que Python se ha vuelto tan popular es su facilidad de uso y aprendizaje. Python es un lenguaje de programación interpretado que tiene una sintaxis clara y concisa. Además, la comunidad de Python es muy activa y proporciona una amplia variedad de documentación, tutoriales y recursos para ayudar a los usuarios a aprender el lenguaje.

En la actualidad, Python se utiliza ampliamente en la industria y en el mundo académico. Muchas empresas, desde startups hasta grandes corporaciones, utilizan Python para el desarrollo de software, el análisis de datos y el aprendizaje automático. Python también es popular en la investigación académica y se utiliza para la programación científica y la simulación.

Referencias Bibliográficas

[1]"¿Qué es Python? | Teradata". Análisis de datos empresariales para un mundo de múltiples nubes | Teradata. https://www.teradata.com/Glossary/What-is-Python (accedido el 12 de marzo de 2023).

[2]"Python Dictionaries". W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_dictionaries.asp (accedido el 12 de marzo de 2023).

[3]A. Tomar. "Listas de Python | 10 operaciones imprescindibles para la manipulación de datos". Medio. https://towardsdatascience.com/python-lists-10-must-know-operations-for-data-manipulation-8ee99fb130a2 (accedido el 12 de marzo de 2023).

[4]"Python Lists". W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_lists.asp (accedido el 12 de marzo de 2023).

[5]"Lista de Python (con ejemplos)". Programiz: aprende a codificar gratis. https://www.programiz.com/python-programming/list (accedido el 12 de marzo de 2023).

[6]"Python Lists - javatpoint". www.javatpoint.com. https://www.javatpoint.com/python-lists (accedido el 12 de marzo de 2023).

[7]"Diccionario Python - GeeksforGeeks". Geeks para Geeks. https://www.geeksforgeeks.org/python-dictionary/ (accedido el 12 de marzo de 2023).

Curso: Tópicos de Ciencia de Datos

Tarea: Investigación Unidad 1

Valor: 20

Fecha de vencimiento: Sunday, 12 de March de 2023, 16:00

Ricardo Mezo Bustamante
mezo21r@gmail.com

Cambiar usuario

7 de 15

Página 1 de 11

Entrega

Enviado para calificar
Calificado
La tarea fue enviada 15 horas 34 minutos antes
El estudiante puede editar esta entrega

Inv_Uni_I_RMB.pdf

Comentarios (0)

Calificación

Calificación:

Hoja de presentación	No contien todos los datos 0 puntos	Datos incompletos 0.5 puntos	Completo 1 puntos	
Introducción	No contiene 0 puntos	Muy pequeña 1.5 puntos	Completa 3 puntos	
Contenido	No cubre los temas 0 puntos	La mitad de los temas 6 puntos	Completo 11 puntos	

Referencias IEEE	No contiene 0 puntos	Una o no tiene el formato 1 puntos	Más de una y formato correcto 2 puntos	
Conclusión	No contien 0 puntos	Muy pequeña 1 puntos	Completa 2 puntos	
Archivo PDF	Sin formato 0 puntos	Correcto 1 puntos		

Calificación actual en el libro de calificaciones

20,00

Comentarios de retroalimentación

Notificar a los estudiantes  Guardar cambios Reiniciar

Materia: Topicos de Ciencias de Datos

Alumno: Ricardo Mezo Bustamante

Unidad I: Python

22 de Febrero de 2023

In [ ]:
```
print("Primer mensaje en Python")
```

```
Primer mensaje en Python
```

Operadores Aritmeticos

Suma, Resta, Multiplicacion, Division

Concatenar usando "comas"

In [ ]:
```
a=10
b=5
print("suma - ",a+b)
print("resta - ",a-b)
print("multiplicacion - ",a*b)
print("division - ",a/b)
print("modulo - ",a%b)
print("division entera",a//b)
```

```
suma -  15
resta -  5
multiplicacion -  50
division -  2.0
modulo -  0
division entera 2
```

suma de dos numeros enteros ingresados por teclado

In [ ]:
```
numero1=int(input("Escribe el primer numero"))
numero2=int(input("Escribe el segundo numero"))
suma=numero1+numero2
print(suma)
```

```
10
```

2.- Multiplicacion, division y residuo

3.- Suma y resta

((2+3)(5*2+5))/2

In [ ]:
```
print(((2+3)*(5**2+5))/2)
```

```
75.0
```

type() permite conocer el tipo de dato

In [ ]:
```
type(a)
```

Out[ ]:
```
int
```

NOTA: Todo lo que se lee es cadena

In [ ]:
```
print('Escribe tu nombre')
name=input()
print(name)
```

```
Escribe tu nombre
ricardo
```

Suma de 2 numeros enteros ingresados por teclado

In [ ]:
```
numero1=int(input("Escribe el primer numero"))
numero2=int(input("Escribe el segundo numero"))
suma=numero1+numero2
print(suma)
print(numero1,'+',numero2,'=',suma)
print(f'{numero1} + {numero2} = {suma}')
print('{} + {} = {}'.format(numero1,numero2,suma))
```

```
101
34 + 67 = 101
34 + 67 = 101
34 + 67 = 101
```

Not Equals: a!=b

Less than: a<b

Less than or equal to: a<=b

Greater than: a>b

Greater than or equal to: a>=b

In [ ]:
```
a=33
b=200
if b>a:
    print("b is greater than a")
```

```
b is greater than a
```

In [ ]:
```
a=33
b=33
if b>a:
    print("b is greater than a")
#else:
    #print("a is greater than b")
elif a==b:
    print('a y b son iguales')
```

```
a y b son iguales
```

Ciclos

While

For

In [ ]:
```
i=1
while i<6:
    print(i)
    i+=1
```

```
1
2
3
4
5
```

In [ ]:
```
i=1
while i<6:
    print(i)
    if i==3:
        break
    i+=1
```

```
1
2
3
```

In [ ]:
```
for x in range(11):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [ ]:
```
for x in 'python':
    print(x)
```

```
p
y
t
h
o
n
```

Listas

En las listas podemos ingresar todo tipos de datos.

Se utilizan los corchetes [] para definir una lista y se separan usando coma.

Las listas son elementos mutables (se pueden cambiar sus valores)

In [ ]:
```
lista=[1,2,3,4,5]
print(lista)
print(lista[2]) #Indice inicia en 0. .. desde izquierda
print(lista[-2]) #Indice inicia en -1 ... desde derecha
```

```
[1, 2, 3, 4, 5]
3
4
```

In [ ]:
```
for x in lista:
    print(x)
```

```
1
2
3
4
5
```

In [ ]:
```
# Cadenas comportamiento similar a las listas, pero son inmutables
palabra="python"
print(palabra[3])
print(palabra[-1])
```

```
h
n
```

In [ ]:
```
# SLICING Seleccion de varios elementos [inicio:final-1]
print(palabra[3:6])
print(palabra[:6])
print(palabra[1:])
print(palabra[1:6:2])
print(palabra[::2])
```

```
hon
python
ython
yhn
pto
```

In [ ]:
```
numeros=[1,2,3,4,5,6,7,8,9,10]
copia=numeros
#type(numeros)
print(numeros)
print(copia)
numeros[5]="cadena"
print(numeros)
print(copia)

# las listas no se copian al asignarlas a otra variable,
# ¡es un puntero a la lista original
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 'cadena', 7, 8, 9, 10]
[1, 2, 3, 4, 5, 'cadena', 7, 8, 9, 10]
```

In [ ]:
```
original=[1,2,3,4,5,6,7,8,9,10]
ccp=original[:] #copia explicita
print(original)
print(ccp)
original[5]=3.34
print(original)
print(ccp)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 3.34, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Indexacion anidada

In [ ]:
```
lespecial=[1,2,3,[4,5,['objetivo']]]
print(lespecial[0])
print(lespecial[3])
print(lespecial[3][2])
print(lespecial[3][2][0])
print(lespecial[3][2][0][:2])
```

```
1
[4, 5, ['objetivo']]
['objetivo']
objetivo
ob
```

Agregar elementos a una lista

list.insert(i,x)

inserta un item x en una posicion i, los otros elementos

despues de la posicion se mueven hacia la derecha

In [ ]:
```
print(lespecial)
lespecial.insert(2,'Me gusta python')
print(lespecial)
```

```
[1, 2, 3, [4, 5, ['objetivo']]]
[1, 2, 'Me gusta python', 3, [4, 5, ['objetivo']]]
```

.append()

Agrega elementos al final de la lista

In [ ]:
```
print(lespecial)
lespecial.append("prueba")
print(lespecial)
```

```
[1, 2, 'Me gusta python', 3, [4, 5, ['objetivo']], 'prueba']
[1, 2, 'Me gusta python', 3, [4, 5, ['objetivo']], 'prueba', 'prueba']
```

In [ ]:
```
frutas=['manzana','banana','cereza','manzana','cereza']
print(frutas)
print(len(frutas))
```

```
['manzana', 'banana', 'cereza', 'manzana', 'cereza']
5
```

In [ ]:
```
lmixta=['abc',342,True,40.2,'male']
print(lmixta)
print(type(lmixta))
```

```
['abc', 342, True, 40.2, 'male']
<class 'list'>
```

.remove()

Elimina el elemento indicado

In [ ]:
```
print(frutas)
frutas.remove('cereza')
print(frutas)
```

```
['manzana', 'banana', 'cereza', 'manzana', 'cereza']
['manzana', 'banana', 'manzana', 'cereza']
```

.pop()

Elimina el elemento indicado por el indice

In [ ]:
```
print(frutas)
frutas.pop(1)
print(frutas)
```

```
['manzana', 'banana', 'manzana', 'cereza']
['manzana', 'manzana', 'cereza']
```

.clear

Limpia (elimina) el contenido de la lista

In [ ]:
```
print(frutas)
frutas.clear()
print(frutas)
```

```
['manzana', 'manzana', 'cereza']
[]
```

.sort()

ordena la lista, ascendente por default

In [ ]:
```
ltexto=['orange','mango','kiwi','piña','banana']
lnumero=[100,50,65,82,43]
ltexto.sort()
print(ltexto)
lnumero.sort()
print(lnumero)
```

```
['banana', 'kiwi', 'mango', 'orange', 'piña']
[43, 50, 65, 82, 100]
```

.sort(reverse=True)

In [ ]:
```
print(ltexto)
ltexto.sort(reverse=True)
print(ltexto)
ltexto.reverse()
print(ltexto)
```

```
['piña', 'orange', 'mango', 'kiwi', 'banana']
['piña', 'orange', 'mango', 'kiwi', 'banana']
['banana', 'kiwi', 'mango', 'orange', 'piña']
```

SETS (Conjuntos)

Se usan para almacenar multiples elementos de una sola variable.

Un conjunto es una coleccion que es desordenado, inmutable y sin indexar.

No permiten valores duplicados.

Sin ordenar significa que los elementos en un conjunto no tienen un orden definido.

Los elementos de un conjunto pueden aparecer en un orden diferente cada vez que los usa, y no

puede ser referido por indice o clave.

In [ ]:
```
cfruta={'manzana','platano','fresa'}
print(cfruta)
type(cfruta)
```

```
{'manzana', 'platano', 'fresa'}
```

Out[ ]:
```
set
```

In [ ]:
```
cfruta={'manzana','platano','fresa','manzana'}
print(cfruta)
```

```
{'manzana', 'platano', 'fresa'}
```

In [ ]:
```
#Conjunto mixto
#Los valores true y 1 se consideran el mismo  valor en conjuntos,
#y son tratados como duplicados
cmixto={"apple","banana","cherry",True,1,2}
print(cmixto)
```

```
{True, 2, 'banana', 'cherry', 'apple'}
```

In [ ]:
```
set1={'abc',34,True,40,'Hombre'}
print(set1)
```

```
{True, 34, 40, 'abc', 'Hombre'}
```

Tupla

Una tupla es una coleccion que se ordena, son inmutables

permiten valores duplicados

In [ ]:
```
tnombre=('Maria','Ramon','Roman','Romeo')
print(tnombre)
type(tnombre)
```

```
('Maria', 'Ramon', 'Roman', 'Romeo')
```

Out[ ]:
```
tuple
```

Diccionario

Estructura de datos con caracteristicas especiales que nos permite almacenar cualquier tipo de valor

Los diccionarios permiten identificar cada elemento mediante una clave (no puede existir dos veces la misma clave)

In [ ]:
```
dPp={'edad':20,'nombre':'Peperrucho','Apellido':'Perez','Estatura':1.70}
print(dPp)
```

```
{'edad': 20, 'nombre': 'Peperrucho', 'Apellido': 'Perez', 'Estatura': 1.7}
```

In [ ]:
```
print(dPp['nombre'])
print(dPp['Estatura'])
```

```
Peperrucho
1.7
```

In [ ]:
```
futbolistas=dict()
futbolistas={1:'Casillas',15:'Ramos',3:'Pique',5:'Puyol',11:'Capdevila',14:'Xavi Al
             8:'Xavi Hernandez',18:'Pedrito',6:'Iniesta',7:'Villa'}
print(futbolistas)
```

```
{1: 'Casillas', 15: 'Ramos', 3: 'Pique', 5: 'Puyol', 11: 'Capdevila', 14: 'Xavi Al
onso', 16: 'Busquets', 8: 'Xavi Hernandez', 18: 'Pedrito', 6: 'Iniesta', 7: 'Vill
a'}
```

In [ ]:
```
for k,v in futbolistas.items():
    print('%d -> %s'%(k,v))
```

```
1 -> Casillas
15 -> Ramos
3 -> Pique
5 -> Puyol
11 -> Capdevila
14 -> Xavi Alonso
16 -> Busquets
8 -> Xavi Hernandez
18 -> Pedrito
6 -> Iniesta
7 -> Villa
```

In [ ]:
```
numElem=len(futbolistas)
print(numElem)
print('\nNumero de elementos del diccionario Futbolistas =%d'%numElem)
```

```
11

Numero de elementos del diccionario Futbolistas =11
```

In [ ]:
```
print(futbolistas.keys())
fKeys=futbolistas.keys()
print('Claves del diccionario futbolistas: \n%s'%fKeys)
```

```
dict_keys([1, 15, 3, 5, 11, 14, 16, 8, 18, 6, 7])
Claves del diccionario futbolistas:
dict_keys([1, 15, 3, 5, 11, 14, 16, 8, 18, 6, 7])
```

In [ ]:
```
fValue=futbolistas.values()
print('Valores del diccionario futbolistas: \n%s'%fValue)
```

```
Valores del diccionario futbolistas:
dict_values(['Casillas', 'Ramos', 'Pique', 'Puyol', 'Capdevila', 'Xavi Alonso', 'B
usquets', 'Xavi Hernandez', 'Pedrito', 'Iniesta', 'Villa'])
```

Obtener valor de un elemento del diccionario mediante su clave

In [ ]:
```
elemento=futbolistas.get(38)
print("el elemento del diccionario con clave 38 es: %s",elemento)
```

```
el elemento del diccionario con clave 38 es: %s None
```

In [ ]:
```
elemento=futbolistas.get(6)
print('el elemento del diccionario con clave 6 es: %s'%elemento)
```

```
el elemento del diccionario con clave 6 es: Iniesta
```

def --Funciones--

You can pass data, know as parameters, into a function

A function can return data as a result.

In Python a function is defined suing the def keyword

In [ ]:
```
def saludo():
    print('Hola desde una funcion python')
saludo()
```

```
Hola desde una funcion python
```

argumentos

In [ ]:
```
def s_arg(fname):
    print('Hola'+fname)
s_arg('Holass')
```

```
HolaHolass
```

In [ ]:
```
def suma(n1,n2):
    add=n1+n2
    print(add)
suma(5,8)
```

```
13
```

In [ ]:
```
def pais(country='Mexico'):
        print('Soy de '+country)
pais('Rusia')
pais('Brasil')
```

```
pais()
pais('India')
```

```
Soy de Rusia
Soy de Brasil
Soy de Mexico
Soy de India
```

Ordenamientos en una funcion

In [ ]:
```
def bienvenida(nombre,apellido):
    print('¡Bienvenido a Python',nombre,apellido,'!')
#Posicional
bienvenida('Pedro','Perez')
#Nominal
bienvenida(apellido='mezo',nombre='ricardo')
```

```
¡Bienvenido a Python Pedro Perez !
¡Bienvenido a Python ricardo mezo !
```

Return

In [ ]:
```
def suma(n1,n2):
    add=n1+n2
    return add
print(suma(5,8))
```

```
13
```

In [ ]:
```
#Area de un triangulo
def area_triangulo(base,altura):
    return base*altura/2
area=area_triangulo(2,3)
print('Area = ',area)
```

```
Area =  3.0
```

Curso: Tópicos de Ciencia de Datos

Tarea: Practicas Unidad 1

Valor: 40

Fecha de vencimiento: Tuesday, 14 de March de 2023, 21:00

Ricardo Mezo Bustamante
mezo21r@gmail.com

Cambiar usuario

7 de 15

Sello (Alt/Shift-Alt/Ctrl-Option + m)

Generando el PDF...

Entrega

Enviado para calificar
Calificado
La tarea fue enviada 11 horas 29 minutos después
El estudiante puede editar esta entrega

Ejercicios_Uni_I_RMB.html

Comentarios (0)

Calificación

Calificación:

Practicas

Hoja de presentación	No contien todos los datos 0 puntos	Datos incompletos 1 puntos	Completo 2 puntos	
Indice	No contiene 0 puntos	Contiene 2 puntos		
Practicas	No contiene 0 puntos	Parcialmente 15 puntos	Todas 25 puntos	Extra clases 30 puntos
Conclusión	No contiene 0 puntos	Pequeña 3 puntos	Completa 6 puntos	

Calificación actual en el libro de calificaciones

40,00

Comentarios de retroalimentación

Notificar a los estudiantes

Guardar cambios

Reiniciar