

EJERCICIOS HOJA 2A

TEMA: TEST CONDICIONALES Y IF STATEMENT

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE SENTENCIAS CONDICIONALES

BIENVENIDA AL USUARIO:

Un ordenador tiene tres usuarios distintos (Alejandro, Naomi y Sergio) y otro usuario invitado. Haz un script que pida el nombre al usuario y le dé una bienvenida personalizada. Crea el script de tal manera que si el usuario no es ninguno de los tres se le dé un saludo genérico. ¿Que ocurre si uno de los usuarios introduce su nombre completamente en minúsculas? ¿Y si lo introduce en mayúsculas? ¿Y si sin querer pone in punto en mitad de su nombre (p.e. nao.mi)? ¿Y si se le cuela una almohadilla (p.e se#rgio)?

CONTRASEÑA SEGURA:

*Por motivos de seguridad una contraseña debe tener una vocal en minúscula, dos símbolos especiales diferentes (pueden ser solo * o #). Dada una contraseña ingresada por el usuario, comprueba si es una contraseña segura e indícalo por pantalla.*

PAR O IMPAR:

Crea un script que dado un número y una potencia compruebe si ese numero elevado a esa potencia es par o impar. (Pista: los números pares tiene resto = 0 al dividirlos por 2)

DIVISION:

Escribir un programa que pida al usuario dos números y muestre por pantalla su división. Si el divisor es cero el programa debe mostrar un error.

LOG-IN:

Crea un script que pida una contraseña al usuario (el script sabe cual es la contraseña correcta). Si la contraseña es correcta el script debe darle la bienvenida al usuario. De lo contrario debe indicarle que la contraseña es incorrecta y darle una segunda oportunidad de introducir la contraseña. Al segundo fallo debe mostrar un mensaje de error y terminar de ejecutarse. Cambia el script para que no distinga entre mayúsculas y minúsculas.

(Pista: Necesitarás in If Statement anidado)

BECAS PARA ESTUDIANTES (BONUS*):

El gobierno quiere otorgar becas de excelencia a los estudiantes con un mínimo de un 8 de media. Además para acceder a la beca el estudiante debe tener entre 17 y 21 años. Crea un script que pida el nombre, la edad y la nota media del estudiante e indique si puede optar a la beca o no.

**Los ejercicios bonus no se resolverán directamente en clase si no que están pensados para que los alumnos los discutan por el chat de Discord y compartan sus soluciones. Las soluciones compartidas de los alumnos se subirán en un archivo a la academia.*

EJERCICIOS HOJA 2B

TEMA: TEST CONDICIONALES Y IF STATEMENT

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE SENTENCIAS CONDICIONALES

EL MAYOR DE CUATRO:

Crea un script que pida al usuario 4 números diferentes y imprima el mayor de los cuatro por pantalla.

RELACION ENTRE NÚMEROS:

Crea un script que pida al usuario 3 números diferentes y le indique si alguno de ellos es la suma de los otros dos.

EL TRIÁNGULO:

En una obra es necesario construir para el tejado de una casa una estructura triangular con tres piezas. El ingeniero se pregunta si dadas la largura de las piezas que ha recibido podrá construir este estructura. Crea un script que dados tres longitudes indique si podría construirse un triangulo con esas piezas.

(Pista: la suma de dos piezas tiene que ser mayor que el lado restante. Esto debe ser así para todas las posibles combinaciones)

BOT DE TRADING:

Un bot de trading está programado para realizar ciertas acciones con respecto a un producto financiero. Crea un script de manera que dado un precio introducido por el usuario, si el precio del producto está por debajo de 100 dólares, el bot imprima por pantalla la orden de comprar. Si está entre 100 y 150 dolores (ambos incluidos) el bot deberá imprimir la orden de hold. Si el precio está estrictamente por encima de 150 el bot deberá imprimir la orden de vender.

MAYUSCULA O MINUSCULA (BONUS*):

Permite que el usuario introduzca una letra (del alfabeto latino) como input. Comprueba si esta es una mayúscula o una minúscula.

**Los ejercicios bonus no se resolverán directamente en clase si no que están pensados para que los alumnos los discutan por el chat de Discord y compartan sus soluciones. Las soluciones compartidas de los alumnos se subirán en un archivo a la academia.*

EJERCICIOS HOJA 2C

TEMA: TEST CONDICIONALES Y IF STATEMENT

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE SENTENCIAS CONDICIONALES ANIDADAS

GRUPOS DE ALUMNOS:

En uno de los cursos se ha dividido a una clase en dos grupos A y B. Para mezclar a los alumnos lo mejor posible se ha asignado a todas las chicas con nombres empezando por la letra E hasta la M en el grupo A y el resto en el B. A los chicos con nombres empezando por la letra A hasta la H y R hasta la Z se les ha asignado al grupo A también, el resto están en el B. Crea un script que pregunte al usuario si es chica o chico y el nombre. El script debe mostrar por pantalla el grupo que le corresponde a ese alumno.

DECLARACION DE LA RENTA:

Para tributar en un determinado país es necesario ser mayor de edad y cobrar más de 1000 euros al mes. Además los tramos impositivos de la renta anual son los siguientes:

RENTA	TIPO IMPOSITIVO
Menos de 15.000 eu	5%
Entre 15.000 y 25.000 eu	15%
Entre 25.000 y 35.000 eu	20%
Entre 35.000 y 60.000 eu	30%
Más de 60.000 eu	45%

Escribe un script que primero compruebe si eres susceptible de que se te aplique algún tipo impositivo y en caso afirmativo imprima por pantalla cuál te tocaría.

RESTAURANTE ONLINE:

En una hamburguesería han abierto la posibilidad de hacer pedidos online. Ofrecen básicamente dos productos de fama mundial: su hamburguesa clásica y la hamburguesa vegana.

Los ingredientes extra de la hamburguesa clásica son:

- Queso Idiazabal
- Bacon
- Huevo

Los ingredientes extra de la hamburguesa vegana son:

- Tofu
- Cebolla caramelizada

Crea un script que le pregunte al usuario que tipo de hamburguesa quiere. En función de la respuesta debe enseñarle los ingredientes extra disponibles y permitirle escoger uno de ellos. Finalmente debe imprimir por pantalla que tipo de hamburguesa se ha elegido y cuales son sus ingredientes.

EJERCICIOS HOJA 3

TEMA: USO DE LISTAS Y BUCLES

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE LISTAS SIMPLES, LISTAS ANIDADAS, SUS FUNCIONES ASOCIADAS Y LAS SENTENCIAS CONDICIONALES Y LOS BUCLES EN EL CONTEXTO DE LAS LISTAS

EJERCICIOS 3A

BUCLES:

1. *Escribe un programa que pida al usuario un número entero y muestre por pantalla una estructura como la de más abajo, donde el valor de entrada es el número de estrellas en el centro de la estructura.*

```
*
**
***
****
*****
****
***
**
*
```

2. *Escribir un programa que almacene la cadena de caracteres **contraseña** en una variable, pregunte al usuario por la contraseña hasta que introduzca la contraseña correcta.*
3. *Crea un script que pida al usuario una palabra y luego muestre por pantalla una a una las letras de la palabra introducida empezando por la última.*
4. *Crea un programa en el que se pregunte al usuario por una frase y una letra, y muestre por pantalla el número de veces que aparece la letra en la frase.*

NUMEROS PRIMOS 1:

Crea un programa que imprima todos los números primos entre el 2 y el 100. Un número primo es un número positivo y entero mayor que uno que no tiene un divisor positivo y entero que no sea 1 o sí mismo.

LISTAS DE CARACTERES:

1. *Crea una lista llamada **frutas** que contengan los siguientes nombres de frutas como cadenas de caracteres: manzana, plátano, cereza, pera, higo, frambuesa y fresa.*
2. *Usa la función `len()` para imprimir la longitud de la lista `frutas`.*
3. *Accede al objeto número 3 de la lista e imprímelo por consola.*
4. *Modifica el segundo objeto de la lista y cámbialo a `mora`.*
5. *Añade el string `mango` al final de la lista.*
6. *Usa el método `insert()` y añade el string `"uva"` al comienzo de la lista.*
7. *Usa un bucle para recorrer la lista e imprimir cada fruta por la consola.*
8. *Usa el método `pop()` para eliminar el último elemento de la lista y guárdalo en una variable llamada `"ultima_fruta"`.*
9. *Realiza un bucle que recorra la lista e imprima cada una de las frutas por consola.*
10. *Modifica el script para que imprima también la longitud de cada nombre de fruta por consola.*
11. *Modifica el script para que recorra la lista de frutas y solo imprima aquellos nombres que tengan más de 5 caracteres.*
12. *Usa el método `remove()` para borrar el string `"cereza"` de la lista.*
13. *Usa el método `clear()` para vaciar la lista.*

Recomendación: En cada paso comprueba que el código hace aquello que quieres

LISTAS NUMERICAS:

1. *Crea una lista llamada "numeros" que contenga los siguientes numeros enteros: [1,2,3,4,5,6,7,8,9,10].*
2. *Crea una nueva lista con los números pares de la lista anterior en orden inverso*
3. *Escribe un bucle que recorra la lista "numeros" e imprima el cuadrado de cada numero por consola.*
4. *Intenta rehacer los pasos 2 y 3 con el menor número de lineas posible (método de compresión).*
5. *Usa un método que te devuelva el número más pequeño de la lista e imprímelo por pantalla*
6. *Haz lo mismo con el número más alto*
7. *Suma todos los elementos de la lista con y sin un bucle.*
8. *Encuentra el índice correspondiente al número 8 en la lista original y en la lista resultante tras el punto 2.*

PILLANDO SOLTURA:

1. *Escribe un programa en Python para encontrar los elementos duplicados de una lista, añadirlos a una nueva lista y borrarlos de la lista. Después imprime una lista con tan solo los elementos únicos.*
2. *Escribe un programa en Python para unir dos listas y ordenarlas en orden ascendente.*
3. *Escribe un script que encuentre el segundo número más grande de una lista.*
4. *Crea un script que cuente el número de elementos más grandes que un determinado número dado por el usuario (supón una lista numérica).*
5. *Crea un script dado un número introducido por el usuario o determinado al inicio del programa, realice la suma de aquellos números que sean divisibles por este.*
6. *Escribe un script que pida un número al usuario y dada una lista encuentre el número más alto que es inferior al número introducido o determinado al inicio del programa.*
7. *Crea un script que extraiga los elementos comunes entre dos listas.*
8. *Crea un script que cuente el número de apariciones de un elemento de una lista en dicha lista (P.e. en la lista lista=[23, 65, 23] el número de apariciones de 23 es 2)*
9. *Escribe un programa que lea una lista de enteros y cree una nueva lista que contenga solo números positivos de la lista original.*
10. *Crea un script que tome una lista de strings y cree una nueva lista que contenga el tamaño de los strings de la lista original.*
11. *Crea un programa que dada una lista de strings, devuelva otra lista con los strings en mayúscula.*

EJERCICIOS 3B

PALABRAS PROHIBIDAS:

Define una lista de 5 palabras aleatorias y una lista de letras prohibidas que contenga tres letras. Filtra las palabras en tu lista original crea una nueva lista de palabras filtradas que solo contenga aquellas palabras que no tienen ninguna letra prohibida.

NUMEROS PRIMOS 2:

Dado una lista de números enteros, escribe un script en Python que devuelva una nueva lista con los números primos de la lista original. Además, el script debe devolver el número total de números primos encontrados y la suma de los números primos encontrados

SCRABBLE:

Supongamos una lista de de caracteres llamada "palabras" que representa una mano de Scrabble. Cada string contiene dos caracteres: el primer carácter es la letra de una ficha y el segundo el numero que representa los puntos de la ficha. Por ejemplo, el string "A5" representa la ficha con la letra A y un valor de 5 puntos. Crea un script que calcule el valor total de los puntos en una mano de scrabble. El valor total será la suma de los puntos de todas las fichas de la mano.

EL COMERCIAL:

Eres un comercial trabajando para una compañía que vende diversos productos. Quieres crear un programa para realizar un seguimiento de los productos que has vendido y el valor total de las ventas. Supongamos que hay un total de 10 productos.

Tú has vendido 5 de estos productos en las siguientes cantidades:

*Producto 1: 3 unidades
Producto 2: 1 unidad
Producto 5: 7 unidades
Producto 6: 2 unidades
Producto 9 : 4 unidades*

Los precios de cada uno de estos productos son como siguen:

<i>Producto 1: 30.0 EU</i>	<i>Producto 6: 44.0 EU</i>
<i>Producto 2: 9.8 EU</i>	<i>Producto 7: 21.2 EU</i>
<i>Producto 3: 42.5 EU</i>	<i>Producto 8: 53.2 EU</i>
<i>Producto 4: 32.6 EU</i>	<i>Producto 9: 25.3 EU</i>
<i>Producto 5: 71.5 EU</i>	<i>Producto 10: 57.8 EU</i>

Crea un script que dada una lista con los productos, sus precios y las unidades vendidas, imprima la cantidad total de ventas, el dinero facturado por producto y el dinero total.

EJERCICIOS 3C

BASE DE DATOS DE UN COLEGIO:

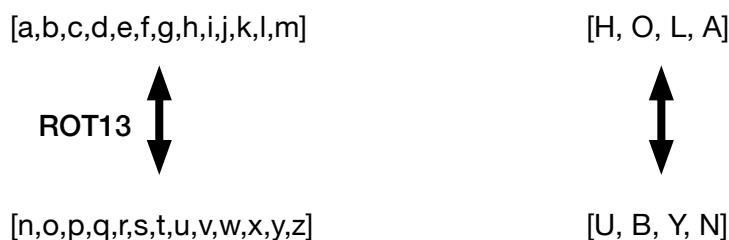
Trabajas en colegio y estas encargado de mantener un seguimiento de las notas de los estudiantes de un clase. En tu base de datos tienes una lista con los nombres de los estudiantes y para cada estudiante debes guardar sus notas provenientes de deberes, exámenes y proyectos. También necesitas calcular a nota media de cada estudiante y la nota media de la clase al completo.

Pista: Para resolver este problema puedes usar una lista anidada donde guardes las notas para cada estudiante. Entonces puedes usar un bucle para recorrer la lista de listas y calcular la nota media de cada estudiante. También puedes usar otro bucle para calcular la nota media de toda la clase.

ENCRIPCIÓN ROT13:

El abecedario latino es un sistema de escritura alfabético más usado del mundo hoy en día. Se compone de 26 letras principales, más ciertas modificaciones y letras adicionales según el idioma del que se trate (por ejemplo, en castellano y gallego se incluye la "ñ", en portugués, francés y catalán la "Ç", en alemán la "ß", etc.).

Aplicar el cifrado ROT13 a un texto se reduce a examinar sus caracteres alfabéticos y sustituirlos por la letra que está 13 posiciones por delante en el alfabeto, volviendo al principio si es necesario y conservando las mayúsculas y minúsculas: a se convierte en n, B se convierte en O, y así hasta la Z, que se convierte en M. Solo quedan afectadas las 26 letras principales que aparecen en el alfabeto latino; los números, símbolos, espacios y otros caracteres se dejan igual.



1. Desarrolla un script que recibiendo de entrada una cadena de caracteres devuelva el texto codificado según el cifrado ROT13
2. Desarrolla ahora un script que compare dos cadenas de caracteres y nos diga si una de ellas esta codificación ROT13 de la otra.

EJERCICIOS HOJA 3

TEMA: USO DE LISTAS Y BUCLES

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE LISTAS SIMPLES, LISTAS ANIDADAS, SUS FUNCIONES ASOCIADAS Y LAS SENTENCIAS CONDICIONALES Y LOS BUCLES EN EL CONTEXTO DE LAS LISTAS

EJERCICIOS 3D

ANÁLISIS DE VENTAS:

Supongamos que eres el propietario de una tienda en línea y tienes una lista de ventas de los últimos 30 días. Quieres analizar las ventas por día de la semana para identificar los días de mayor venta.

Pista 1: Puedes crear dos listas, una con las ventas por cada día del mes como por ejemplo...
ventas = [120, 80, 140, 200, 75, 100, 180, 220, 160, 110, 90, 120, 170, 190, 250, 300, 95, 110, 140, 180, 200, 160, 120, 80, 170, 150, 210, 190, 230, 250]

Y otra lista con los días de la semana:

días_semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]

Después puedes crear una nueva lista con una entrada por cada día de la semana y usar un bucle para añadir a esta lista la suma de las ventas correspondientes a cada uno de los días de la semana.

Pista 2: Puede que necesites una variable que lleve la cuenta del día de la semana actual y se reinicie a cero cuando llegue al séptimo día.

VALIDAR ACCESO A UN SITIO WEB:

Supongamos que eres un administrador de sistemas y necesitas validar el acceso de los usuarios a un sitio web. Crea un script que verifique si el nombre de usuario y la contraseña ingresados son correctos y permita el acceso solo si ambos son correctos.

Pista 1: Puedes crear dos listas, una con los nombre de usuario como por ejemplo...

nombres_usuario = ["juan123", "ana456", "pedro789"]

Y otra lista con las contraseñas guardadas para cada usuario...

contraseñas = ["clave123", "clave456", "clave789"]

Otra opción puede ser que crees una lista de listas con la forma:

nombres_contraseñas = [["juan123", "clave123"], ["ana456", "clave456"], ["pedro789", "clave789"]]

Después puedes pedir el usuario y contraseña y comprobar si coinciden.

Pista 2: Para verificar si el usuario y contraseña son correctos puedes crear un bucle donde recorras los nombres de usuario y compruebes con un if si el nombre de usuario introducido y la contraseña coinciden con los datos de tus listas.

EJERCICIOS HOJA 3 BONUS

TEMA: USO DE LISTAS Y BUCLES

[Estos ejercicios son para aquellos que ya hayan terminado con las hojas de ejercicios disponibles para el tema 3].

MATRIZ:

Crea un script que dada una lista de listas M (numérica), identifique si se trata de una matriz y en ese caso imprima dos listas correspondientes a:

1. La fila cuyos elementos suman el máximo
2. La columna cuyos elementos suman el máximo

Si no se trata de una matriz devolverá dos listas vacías.

Por ejemplo:

$M1 = [[2,5,3],[6,1,8],[7,5,4]]$ devolverá: $L1 = [7,5,4]$ y $L2 = [2,6,9,7]$

$M2 = [[4,2,3],[4,5],[6,8,2]]$ devolverá: $L1 = []$ y $L2 = []$

(Nota: Definimos matriz como una lista de listas donde todas las listas internas tienen el mismo número de objetos)

STRING A LISTA:

Desarrolla un script en Python que dado una cadena de caracteres con la siguiente información: nombre, apellido, DNI, código_asignatura, convocatoria, nota1, nota2, nota3 ... Por ejemplo: David Fernandez 12311267A 43527 2 2.1 4.6 3.4. El script debe crear una lista con esos datos, introducirlo en una lista de listas donde se encuentra la información de todos los alumnos e imprimir la nota media de los alumnos junto con el DNI.

Supón ahora que tu input es un string como este:

"David Fernandez 12311267A 43527 2 9.1 7.6 2.4\n

Maria Garcia 12316487A 43527 2 7.1 8.6 5.4\n

Juan Perez 647829236A 43527 2 8.1 8.5 8.4\n,"

Reescribe el script para que procese ese input adecuadamente e imprima la nota media y el DNI de todos los alumnos en ese string.

EJERCICIOS HOJA 4

TEMA: USO DE ARRAYS

OBJETIVO: FAMILIARIZACIÓN CON LA CREACION DE ARRAYS CON NUMPY Y SUS MÉTODOS ASOCIADOS

EJERCICIOS 4A

INSTALACION DE NUMPY:

1. Instala numpy dentro de tu **environment de trabajo** usando pip o conda.
2. Comprueba que numpy ha sido instalado correctamente y la versión instalada (puedes usar `conda search package_name --info`)

ARRAYS 1D PARTE 1:

1. Crea un `array_1` lleno de ceros con una longitud de 8 elementos.
2. Haz que todos los elementos de este array sean igual a 2
3. Crea un `array_2` que contenga todos los números pares del 1 al 10.
4. Suma todos los elementos del `array_2` usando un bucle y después usando un método de *numpy*. Compara los resultados
5. Revierte `array_2` y guárdalo en una variable independiente.
6. Encuentra los elementos comunes entre `array_1` y `array_2` y entre `array_2` y `array_2_revertido`
(Pista: Investiga el uso de `intersect1d()` de *numpy*)
7. Crea un array lleno de 1s con una longitud dada por el usuario

ARRAYS 1D PARTE 2:

1. Crea un array con 15 números enteros aleatorios entre 1 y 100
2. Multiplica todos los elementos del array usando un bucle y después usando un método de *numpy*. Compara los resultados
3. Crea otro array con 15 números decimales aleatorios entre 0 y 1
4. Suma los elementos de ambos arrays elemento por elemento. Resuélvelo usando un operador y después con una función de *numpy*
(Pista: busca en google que función de *numpy* hace esto)
5. Ahora réstalos. Resuélvelo usando un operador y después con una función de *numpy*
(Pista: busca en google que función de *numpy* hace esto)

6. Haz lo mismo con la multiplicación elemento por elemento. Usa un operador y después con una función de *numpy*
(Pista: busca en google que función de *numpy* hace esto)
7. Encuentra el valor más alto del primer array que has creado.
8. Calcula la media (mean), la mediana (median) y la desviación estándar (standard deviation) de los arrays (Nota: No nos importa el significado matemático de estos valores, lo importante es que encuentres que función de numpy necesitas. Puedes hacer la búsqueda en castellano o en inglés, aunque en inglés muchas veces suele haber más resultados).

ARRAYS 2D

9. Crea un array lleno de 1s con una longitud dada por el usuario
10. Cambia la forma del array para que tenga una estructura de tipo (filas, columnas)
11. Crea una “matriz identidad” con la misma forma que el array anterior (filas, columnas)
12. Concatena ambas estructuras horizontalmente y verticalmente
(Pista: Investiga el funcionamiento de `concatenate()` y de `vstack()` y `hstack()` de numpy)

EJERCICIOS HOJA 4

TEMA: USO DE ARRAYS

OBJETIVO: FAMILIARIZACIÓN CON LA CREACION DE ARRAYS CON NUMPY Y SUS MÉTODOS ASOCIADOS

EJERCICIOS 4B

CALCULO DE NOTAS FINALES

Supongamos que tienes un conjunto de calificaciones de un grupo de estudiantes en un curso. Cada estudiante tiene cuatro calificaciones: dos exámenes, un trabajo final y una participación en clase. Quieres calcular la nota final de cada estudiante, donde los exámenes valen un 30% cada uno, el trabajo final vale un 30% y la participación en clase vale un 10%. Para ello, puedes usar NumPy para crear un array de 4 columnas y n filas, donde n es el número de estudiantes. Cada columna representa una de las calificaciones y cada fila representa un estudiante. Luego, puedes usar operaciones de NumPy para calcular la nota final de cada estudiante y almacenarla en un nuevo array de una sola columna.

ANALISIS DE DATOS - VENTAS POR MES

Supongamos que tienes un conjunto de datos de ventas de una tienda durante un año. Cada fila representa una venta y tiene tres columnas: la fecha de la venta, el monto de la venta y la categoría de producto vendido (por ejemplo, electrónicos, ropa, alimentos, etc.). Quieres analizar estos datos para determinar cuánto fue el monto total de ventas en cada mes. Para ello, puedes usar NumPy para cargar los datos en un array de 3 columnas y n filas, donde n es el número de ventas. Luego, puedes usar operaciones de NumPy para filtrar los datos por mes y sumar los montos de venta correspondientes.

(Pista 1) Tu array de entrada puede tener un a forma de este tipo:

```
# Datos de ventas de la tienda
ventas = np.array([
    ['2022-01-01', 100, 'ropa'],
    ['2022-01-02', 200, 'alimentos'],
    ['2022-01-03', 150, 'ropa'],
    ['2022-02-01', 120, 'alimentos'],
    ['2022-02-02', 180, 'electrónicos'],
    ['2022-02-03', 200, 'alimentos'],
    ['2022-03-01', 90, 'ropa'],
    ['2022-03-02', 110, 'electrónicos'],
    ['2022-03-03', 100, 'alimentos']
])
```

(Pista 2: puedes cambiar el tipo de dato del array de string a entero usando `array[:,i].astype(int)`)

ANÁLISIS DE DATOS CLIMÁTICOS

Supongamos que tienes un conjunto de datos de clima que contiene información sobre la temperatura, la humedad y la presión atmosférica en una ciudad durante un año. Quieres analizar estos datos para determinar cuál fue la temperatura promedio de cada mes, cuál fue la humedad promedio y la presión atmosférica promedio durante todo el año. Para ello, puedes usar NumPy para cargar los datos en un array de 3 columnas y n filas, donde n es el número de mediciones. Luego, puedes usar operaciones de NumPy para filtrar los datos por mes y calcular las medias de temperatura, humedad y presión atmosférica correspondientes.

(Pista 1) Tu array de entrada podría ser algo como esto, con datos de temperatura, humedad, presión y mes del año:

```
# Datos de clima
clima = np.array([
    [20, 70, 1009, 1],
    [21, 60, 1011, 1],
    [22, 40, 1010, 1],
    [18, 75, 1012, 2],
    [21, 60, 1008, 3],
    [22, 65, 1008, 3],
    [25, 60, 1010, 4],
    [27, 49, 1007, 5],
    [29, 50, 1007, 5],
    [28, 51, 1007, 5],
    [30, 45, 1005, 6],
    [10, 30, 1005, 6],
    [32, 40, 1002, 7],
    [33, 35, 1001, 8],
    [31, 45, 1003, 9],
    [30, 42, 1001, 9],
    [29, 42, 1002, 9],
    [35, 43, 1001, 9],
    [28, 50, 1006, 10],
    [25, 60, 1010, 11],
    [27, 59, 1012, 11],
    [24, 58, 1011, 11],
    [22, 70, 1011, 12]
])
```

EJERCICIOS HOJA 4

TEMA: USO DE ARRAYS

OBJETIVO: FAMILIARIZACIÓN CON LA CREACION DE ARRAYS CON NUMPY Y SUS MÉTODOS ASOCIADOS

EJERCICIOS 4C

DATOS CINEMATográfICOS

Supongamos que tienes un conjunto de datos de películas que contiene información sobre su título, género, duración, año de lanzamiento y calificación. Quieres analizar estos datos para determinar cuál es el género de película más popular, cuántas películas se lanzaron en cada década y cuál es la duración promedio de cada género de película.

(Pista 1: Tu array de entrada puede tener la forma...)

```
# array con datos de peliculas
peliculas = np.array([
    ['Peli 1', 'Comedia', 120, 1990, 8.5],
    ['Peli 2', 'Acción', 110, 2005, 7.8],
    ['Peli 3', 'Drama', 95, 2010, 6.9],
    ['Peli 4', 'Comedia', 100, 1985, 7.5],
    ['Peli 5', 'Acción', 130, 2015, 8.1],
    ['Peli 6', 'Drama', 115, 2000, 6.7],
    ['Peli 7', 'Comedia', 90, 1995, 8.2],
    ['Peli 8', 'Acción', 105, 2010, 7.4],
    ['Peli 9', 'Drama', 125, 1980, 6.8],
    ['Peli 10', 'Comedia', 95, 2000, 8.0]
])
```

(Pista 2: puede ser útil investigar `np.unique`, `np.argsort` y `np.count_nonzero`)

EMPRESA DE ELECTRONICA

Supongamos que trabajas en una empresa que fabrica dispositivos electrónicos y quieres analizar los datos de calidad de los componentes utilizados en la producción de dichos dispositivos. Tienes un conjunto de datos que contiene información sobre la fecha de producción, el tipo de componente, el lote al que pertenece el componente y la puntuación de calidad del componente (un número entre 0 y 100). Quieres analizar estos datos para determinar cuál es el tipo de componente con la puntuación de calidad más alta, cuántos componentes se produjeron en cada mes y cuál es la puntuación de calidad promedio de cada tipo de componente.

(Pista 1: Tu array de entrada puede tener la forma...)

```
# Crear un array con los datos
datos = np.array(['2022-01-01', 'Componente 1', 'Lote A', 80],
                  ['2022-01-15', 'Componente 1', 'Lote B', 90],
                  ['2022-02-01', 'Componente 2', 'Lote C', 85],
                  ['2022-02-15', 'Componente 2', 'Lote D', 95],
                  ['2022-03-01', 'Componente 1', 'Lote E', 75],
                  ['2022-03-15', 'Componente 2', 'Lote F', 90])
```

(Pista 2: puede ser útil investigar np.unique y np.argmax)

EJERCICIOS HOJA 5

TEMA: TUPLAS Y SETS

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE TUPLAS Y SETS, SUS CARACTERÍSTICAS Y METODOS ASOCIADOS

EJERCICIOS 5A

TRABJANDO CON TUPLAS:

1. Crea una tupla con tres elementos y imprime por pantalla cada uno de ellos en una nueva línea.
2. Crea una lista con tres elementos e intenta modificarla. Haz lo mismo con la tupla. ¿Cuáles son las diferencias?
3. Crea una tupla de enteros y devuelve la suma de los elementos.
4. Crea un script que dada una tupla que contiene strings cree una nueva tupla con el primer caracter de cada string.
5. Crea un script que dada una tupla de números devuelva el producto de todos los números pares.
6. Crea un script que dada una tupla de números, devuelva la tupla con los numeros ordeandos en orden descendente.
7. Crea un script que dada una tupla con números enteros repetidos, elimine los duplicados. (Puedes usar sets).
8. Crea un script que dada una tupla y un numero entero, devuelve verdadero si el numero se encuentra en la tupla y falso en el caso contrario.
9. Crea un script que dadas dos tuplas cree una tupla resultante de la union de ambas.
10. Crea un script que dada una tupla de números devuelva e máximo y el mínimo.
11. Crea un script que dada una tupla con strings devuelva el string más largo y el más corto. (Prueba añadiendo key=len a las funciones max y min).
12. Crea un script que dada una tupla devuelva el contenido en orden revertido.
13. Crea un script que dada una tupla de tuplas, donde cada tupla interna contiene dos elementos, devuelva una nueva tupla en la que cada elemento sea la suma de los dos elementos de la tupla interna correspondiente.

TRBAJANDO CON SETS:

14. Crea un set y elimina uno de sus elementos.
15. Crea un set vacío.
16. Crea dos sets y encuentra su union, su intersección y su diferencia.
17. Crea un script que dados dos sets cree uno nuevo que contenga solo los elementos comunes de ambos.
18. Crea un script que dado un set con números devuelva el numero máximo y mínimo.
19. Crea un script que dados dos sets cree uno nuevo solo con los elementos únicos de cada uno de los sets.
20. Crea un set con colores y comprueba si cierto color se encuentra en el set.
21. Crea un script que dados dos sets cree un nuevo set con los elementos que están en el primer set pero no en el segundo.
22. Crea un script que dado un set de enteros devuelva el producto de todos los números dentro del set.

EJERCICIOS HOJA 5

TEMA: TUPLAS Y SETS

OBJETIVO: APLICACIÓN DE TUPLAS Y SETS, SUS CARACTERÍSTICAS Y METODOS ASOCIADOS EN EL MUNDO REAL

EJERCICIOS 5B

RED SOCIAL:

Una red social tiene una base de datos de usuarios y sus correspondientes amistades. Crea un programa que tome una base de datos de una red social como una lista de tuplas, donde cada tupla contiene el nombre del usuario y una lista de sus amigos. Los nombres repetidos en la lista de amigos corresponden a usuarios con varias cuentas diferentes. Deberas eliminar las cuentas duplicadas y después devolver una tupla de tuplas que contiene el número real de amigos por usuario y el usuario con más amigos.

Pista 1: Tus datos de entrada podrían ser así —> red_social = [("Juan", ["Maria", "Pedro", "Luis"]), ("Maria", ["Juan", "Pedro", "Juan"]), ("Pedro", ["Juan", "Maria"]), ("Luis", ["Juan"])]

Pista 2: Para eliminar duplicidades puedes usar sets

LA BIBLIOTECA:

Una biblioteca tiene una lista de libros y sus autores. Crea un programa que tome la lista de libros y sus autores como una lista de tuplas, donde cada tupla contiene el título del libro y el nombre del autor, y devuelva una nueva lista de tuplas que contenga el título del libro y el apellido del autor.

Pista: Tus datos de entrada podrían ser así —> lista_libros = [('El aleph', 'Jorge Luis Borges'), ('Cien años de soledad', 'Gabriel Garcia Márquez'), ('La ciudad y los perros', 'Mario Vargas Llosa')]

DATOS CLIENTES:

Una compañía tiene dos bases de datos de clientes. La primera base de datos contiene el nombre del cliente, la dirección de correo electrónico y el número de teléfono. La segunda base de datos contiene el nombre del cliente, la dirección y el historial de pedidos. Escribe un programa que tome las dos bases de datos como listas de tuplas y devuelva una nueva lista de tuplas que contenga solo los clientes que aparecen en ambas bases de datos. Los clientes se consideran iguales si tienen el mismo nombre.

Pista: Tus datos de entrada podrían ser así —>
base_datos1 = [("Juan", "juan@example.com", "555-1234"), ("Maria", "maria@example.com", "555-5678"), ("Pedro", "pedro@example.com", "555-9012")]
base_datos2 = [("Juan", "Calle 123", ["Libro1", "Libro2"]), ("Maria", "Calle 456", ["Libro3"]), ("Luis", "Calle 789", ["Libro4"])]

EJERCICIOS HOJA 6

TEMA: DICCIONARIOS

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE DICCIONARIOS, SUS CARACTERÍSTICAS Y METODOS ASOCIADOS

EJERCICIOS 6A

TRABAJANDO CON DICCIONARIOS:

1. Crea un diccionario vacío llamado "mi_diccionario".
2. Agrega un par clave-valor a "mi_diccionario" donde la clave sea "nombre" y el valor sea tu nombre.
3. Accede e imprime el valor asociado con la clave "nombre" en "mi_diccionario".
4. Verifica si la clave "edad" existe en "mi_diccionario". Imprime "True" si existe y "False" en caso contrario.
5. Crea un diccionario llamado "estudiante" con los siguientes pares clave-valor: "nombre" con el nombre del alumno, "edad" con su edad y "materia" con su materia favorita.
6. Actualiza el valor de la clave "edad" en el diccionario "estudiante" para reflejar la edad actual de tu amigo.
7. Elimina el par clave-valor con la clave "materia" del diccionario "estudiante".
8. Imprime todas las claves en el diccionario "estudiante".
9. Crea un diccionario llamado "agenda" con tres entradas: "Juan" con el valor "1234567890", "Joana" con el valor "9876543210" y "Jimena" con el valor "5555555555".
10. Agrega una nueva entrada al diccionario "agenda" con la clave "Julio" y el valor "9998887777".
11. Imprime el número de entradas (pares clave-valor) en el diccionario "agenda".
12. Crea una lista llamada "claves" que contenga todas las claves del diccionario "agenda".
13. Verifica si la clave "Juan" existe en el diccionario "agenda". Imprime "True" si existe y "False" en caso contrario.
14. Elimina la entrada con la clave "Jimena".
15. Utiliza un bucle for para iterar sobre todas las claves en el diccionario "agenda" e imprime cada par clave-valor en el formato "Nombre: Número".

16. Utiliza el método "get()" para obtener el valor asociado con la clave "Juan" en el diccionario "agenda". Si la clave no existe, imprime "Clave no encontrada".
17. Borra todas las entradas del diccionario "agenda".

LISTAS DE DICCIONARIOS:

18. Crea una lista llamada "estudiantes" que contenga dos diccionarios. Cada diccionario representa a un estudiante y tiene las claves "nombre" y "edad" con sus respectivos valores. Recorre la lista e imprime el nombre y edad de cada estudiante.
19. Agrega un nuevo estudiante a la lista "estudiantes" utilizando un diccionario con las mismas claves "nombre" y "edad". Imprime la lista actualizada.
20. Elimina el segundo estudiante de la lista "estudiantes". Imprime la lista actualizada.
21. Actualiza la edad del primer estudiante en la lista "estudiantes" a un nuevo valor. Imprime la lista actualizada.

ANIDAMIENTO DE DICCIONARIOS:

22. Crea un diccionario llamado "productos" que contenga dos entradas. Cada entrada representa un producto y tiene a su vez las claves "nombre" y "precio" con sus respectivos valores. Recorre el diccionario e imprime el nombre y precio de cada producto.
23. Agrega un nuevo producto al diccionario "productos" utilizando una nueva clave y valor. Imprime el diccionario actualizado.
24. Crea un diccionario llamado "equipos" que contenga tres entradas. Cada entrada representa un equipo deportivo y tiene las claves "nombre" y "jugadores" con sus respectivos valores. Los valores de "jugadores" deben ser listas con los nombres de los jugadores. Recorre el diccionario e imprime el nombre del equipo y la lista de jugadores.
25. Agrega un nuevo equipo al diccionario "equipos" utilizando una nueva clave y valor. La lista de jugadores debe contener al menos tres nombres. Imprime el diccionario actualizado.
26. Actualiza la lista de jugadores de uno de los equipos existentes en el diccionario "equipos". Agrega un nuevo jugador a la lista. Imprime el diccionario actualizado.

EJERCICIOS HOJA 6

TEMA: DICCIONARIOS

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE DICCIONARIOS, SUS CARACTERÍSTICAS Y METODOS ASOCIADOS

EJERCICIOS 6B

REGISTRO DE VENTAS:

Tienes una tienda y deseas realizar un seguimiento de las ventas diarias de tus productos. Cada producto tiene un nombre y una cantidad vendida. Implementa un programa en Python que utilice un diccionario para almacenar la información de las ventas. El programa debe permitir registrar las ventas de productos, actualizar la cantidad vendida de un producto existente y calcular el total de ventas diarias.

(Pista: puedes comenzar con un diccionario vacío e ir añadiendo cada producto)

ADMINISTRACION DE PROYECTOS:

Eres un gerente de proyectos y necesitas un programa para administrar las tareas y responsabilidades de tu equipo. Cada tarea tiene un nombre, una descripción y un responsable asignado. Implementa un programa en Python que utilice un diccionario para almacenar la información de las tareas. El programa debe permitir agregar nuevas tareas, asignar responsables a las tareas existentes, actualizar las descripciones de las tareas y mostrar la lista completa de tareas y responsables.

(Pista: puedes comenzar con un diccionario vacío y construir un diccionario de diccionarios)

REGISTRO DE ASISTENCIAS:

Eres un profesor y deseas realizar un seguimiento de la asistencia de tus estudiantes a lo largo del semestre. Cada estudiante tiene un nombre y una lista de fechas en las que asistió a clases. Implementa un programa en Python que utilice un diccionario para almacenar la información de las asistencias. El programa debe permitir registrar la asistencia de los estudiantes, agregar nuevas fechas de asistencia y mostrar la lista de estudiantes y las fechas en las que asistieron.

(Pista: puedes comenzar con un diccionario vacío y construir un diccionario de listas)

REGISTRO DE PUNTAJES:

Implementa un programa en Python que permita registrar y mantener un seguimiento de los puntajes de un juego. El programa debe permitir a los jugadores ingresar sus nombres y puntajes, almacenarlos en un diccionario y proporcionar funcionalidades para mostrar el puntaje más alto, el promedio de puntajes y la cantidad total de jugadores.

EJERCICIOS HOJA 6

TEMA: DICCIONARIOS

OBJETIVO: FAMILIARIZACIÓN CON EL USO DE DICCIONARIOS, SUS CARACTERÍSTICAS Y METODOS ASOCIADOS

EJERCICIOS 6C

GESTIÓN DE EMPLEADOS:

Imagina que eres el gerente de recursos humanos de una empresa y necesitas gestionar la información de los empleados. Cada empleado tiene un nombre, salario y departamento al que pertenece. Implementa un programa en Python que permita agregar nuevos empleados, actualizar el salario de un empleado existente, mostrar la lista completa de empleados y calcular el promedio salarial por departamento.

ANÁLISIS DE VOTACIONES:

Supongamos que tienes los resultados de una elección con los nombres de los candidatos y la cantidad de votos obtenidos por cada uno. Implementa un programa en Python que permita registrar los votos, mostrar la lista completa de candidatos y sus votos, encontrar al candidato ganador (con más votos) y calcular el porcentaje de votos que obtuvo cada candidato.

EJERCICIOS HOJA 7

TEMA: FUNCIONES

OBJETIVO: FAMILIARIZACIÓN CON EL USO EL USO DE PARAMETROS Y ARGUMENTOS.

1. Define una función llamada "saludar" que tome un parámetro "nombre" y muestre un saludo personalizado.
2. Crea una función llamada "suma" que tome dos parámetros "a" y "b" e imprima la suma de ambos.
3. Escribe una función llamada "calcular_area_rectangulo" que tome dos parámetros "base" y "altura" y calcule el área de un rectángulo.
4. Define una función llamada "imprimir_lista" que tome una lista como parámetro y la imprima en la consola.
5. Crea una función llamada "es_par" que tome un número como parámetro e imprima True si es par, o False si es impar.
6. Escribe una función llamada "concatenar_strings" que tome dos parámetros "cadena1" y "cadena2" e imprima la concatenación de ambas cadenas.
7. Define una función llamada "obtener_maximo" que tome una lista de números como parámetro y devuelva el número máximo de la lista.
8. Crea una función llamada "convertir_fahrenheit_a_celsius" que tome un parámetro "fahrenheit" y devuelva su equivalente en grados Celsius.
9. Escribe una función llamada "calcular_edad" que tome dos parámetros: "año_actual" y "año_nacimiento" y calcule la edad de una persona.
10. Define una función llamada "es_divisible" que tome dos parámetros "num" y "divisor" e imprima True si "num" es divisible por "divisor", o False si no lo es.
11. Crea una función llamada "mostrar_info_persona" que tome tres argumentos de palabra clave: "nombre", "edad" y "ciudad". La función debe imprimir en la consola la información de una persona en un formato legible.

12. Escribe una función llamada "calcular_promedio" que tome una lista de números como parámetro y calcule el promedio de esos números. Si no se proporciona una lista, debe usar una lista vacía por defecto.
13. Crea una función llamada "calcular_potencia" que tome dos parámetros "base" y "exponente", y calcule la potencia de la base elevada al exponente. Utiliza 2 como valor por defecto para el exponente.
14. Define una función llamada "imprimir_info_alumno" que tome un argumento posicional "nombre" (y sin valor por defecto) y varios argumentos de palabra clave: "edad", "curso" y "promedio" (puedes ponerles como valor por defecto None). La función debe imprimir la información del alumno en un formato legible.

EJERCICIOS HOJA 7

TEMA: FUNCIONES

OBJETIVO: FAMILIARIZACIÓN CON EL USO EL USO DE RETURN Y MODULOS.

VALIDACIÓN DE FORMULARIO

Crea un programa que valide un formulario de registro. Crea una función llamada **validar_formulario** que reciba diferentes campos de un formulario (nombre, correo electrónico y número de teléfono) y verifique si los valores ingresados cumplen con los requisitos especificados, siendo estos:

1. Que el nombre tenga una longitud mínima de 3 caracteres
2. Que el teléfono este conformado por dígitos y tenga una longitud de 9 caracteres
3. Que el email contenga un "@" y un "."

GESTIÓN DE VENTAS

Crea un programa que permita gestionar las ventas de una tienda. Utiliza una estructura de datos adecuada para almacenar la información de las ventas (por ejemplo, una lista de diccionarios). Implementa dos funciones, una para agregar el producto vendido con su precio y otro para mostrar las ventas de productos con sus respectivos precios.

(La base de datos puede tener la forma [{"Producto": producto1, "Precio": precio1}, {"Producto": producto2, "Precio": precio2}...])

CONTRASEÑA SEGURA

Crea un script que solicite una contraseña, analice si es segura y si no lo es sugiera una nueva contraseña. Para ello puedes crear un script *validador.py* que contenga una función *validar_contraseña* que reciba una cadena y verifique si cumple con los requisitos mínimos de una contraseña segura (por ejemplo, longitud mínima, presencia de letras mayúsculas, letras minúsculas, números y caracteres especiales). La función debe devolver un valor booleano que indique si la contraseña es válida o no. Por otro lado puedes crear otro script *creador.py* con una función llamada *generar_contraseña* que genere contraseñas seguras de forma aleatoria. La

función debe permitir especificar la longitud de la contraseña y qué tipos de caracteres deben incluirse (por ejemplo, letras mayúsculas, letras minúsculas, números y caracteres especiales).

(Para el generador de contraseñas puedes probar a usar los módulos random y string)

PRACTICA 7C

TEMA: FUNCIONES RECURSIVAS

FACTORIAL

Implementa una función recursiva llamada factorial que calcule el factorial de un número entero positivo. El factorial de un número n se define como el producto de todos los números enteros positivos desde 1 hasta n .
(El factorial de 0 y de 1 es igual a 1)

POTENCIA

Implementa una función recursiva llamada potencia que calcule el resultado de elevar un número a una potencia dada. Puedes asumir que tanto el número como la potencia son enteros no negativos.

SUMA ELEMENTOS DE UNA LISTA

Crea una función recursiva llamada suma_lista que calcule la suma de todos los elementos de una lista de enteros. Puedes asumir que la lista no está vacía.

NUMERO TRIANGULAR

Crea una función recursiva llamada numero_triangular que calcule el n -ésimo número triangular. Un número triangular se obtiene al sumar todos los números naturales desde 1 hasta n .

EJERCICIOS PYTHON AVANZADO

HOJA 2

TEMA: MANEJO DE ARCHIVOS Y EXCEPCIONES

SUMA Y VALUEERROR

Un problema común al solicitar una entrada numérica ocurre cuando las personas ingresan texto en lugar de números. Cuando intentas convertir la entrada a un entero (int), obtendrás un ValueError. Escribe un programa que solicite dos números. Suma los números y muestra el resultado. Captura el ValueError si alguno de los valores de entrada no es un número e imprime un mensaje de error amigable. Prueba tu programa ingresando dos números y luego ingresando texto en lugar de un número. Envuelve tu código del en un bucle while para que el usuario pueda continuar ingresando números incluso si comete un error ingresando texto en lugar de un número.

MANIPULACION DE ARCHIVOS Y FILENOTFOUNDERROR

Crea dos archivos, cats.txt y dogs.txt. Almacena al menos tres nombres de gatos en el primer archivo y tres nombres de perros en el segundo archivo. Escribe un programa que intente leer estos archivos e imprima el contenido de cada archivo en la pantalla. Envuelve tu código en un bloque try-except para capturar el error de FileNotFoundError, e imprime un mensaje amigable si falta algún archivo. Mueve uno de los archivos a una ubicación diferente en tu sistema y asegúrate de que el código en el bloque except se ejecute correctamente. Modifica tu bloque except para que falle en silencio si falta alguno de los archivos.

PALABRAS COMUNES

Encuentra o crea algunos textos que te gustaría analizar (puedes visitar Project Gutenberg (<http://gutenberg.org/>) o crear textos usando ChatGPT). Copia el texto sin formato desde tu navegador en un archivo de texto en tu computadora (o descarga los archivos). Averigua cuántas veces aparece una palabra o frase en el texto (puedes usar el método count()).

BUSCANDO EN PI

Busca si tu fecha de nacimiento esta en los primeros 10000 digitos de pi (y en que posición. Puedes usar `find()`).
Puedes usar el archivo `pi_10000.txt`

NUMERO FAVORITO

Escribe un programa que solicite al usuario su número favorito. Utiliza `json.dump()` para almacenar este número en un archivo. Escribe un programa separado que lea este valor e imprima el mensaje: "Sé cuál es tu número favorito... Es ____." Combina ambos programas en un solo archivo (puedes crear tantas funciones como necesites). Si el número ya está almacenado, muestra el número favorito al usuario. Si no lo está, solicita al usuario su número favorito y guárdalo en un archivo. Ejecuta el programa al menos dos veces para asegurarte de que funciona correctamente.

REFACTORIZACIÓN:

Revisa los ejercicios del modulo "Python para Principiantes". ¿Hay algún ejercicio que pudiese dividirse en funciones? ¿Y alguno que podría optimizarse usando bloques `try-except`? Si es así reescríbelos usando estas estructuras.

EJERCICIOS PYTHON AVANZADO

HOJA 3A

TEMA: MANEJO DE CLASES Y OBJETOS

CLASE PERSONA

Define una clase Persona con atributos como nombre, edad y profesión. Luego, crea varios objetos de esta clase y muestra su información.

CALCULADORA BÁSICA

Crea una clase llamada "Calculadora" con métodos para sumar, restar, multiplicar y dividir. Crea objetos de esta clase y realiza algunas operaciones básicas.

LIBRO

Crea una clase "Libro" con atributos como título, autor y año de publicación. Luego, crea varios objetos Libro y muestra su información.

RECTÁNGULO

Crea una clase "Rectangulo" con atributos de longitud y ancho. Implementa un método para calcular el área y el perímetro del rectángulo.

DADO

Crea una clase "Dado" que simule el lanzamiento de un dado de 6 caras. Implementa un método para lanzar el dado y mostrar el resultado (quizás te convenga usar el modulo *random*).

COCHE

Crea una clase "Coche" con atributos como marca, modelo y año. Implementa un método para encender el coche y otro para apagarlo (puedes simular el encendido y apagado con una variable booleana).

EJERCICIOS PYTHON AVANZADO

HOJA 3B

TEMA: MANEJO DE CLASES Y OBJETOS

CUENTA BANCARIA

Crea una clase "CuentaBancaria" con atributos como número de cuenta y saldo. Implementa métodos para depositar y retirar dinero, y muestra el saldo actual.

LISTA DE TAREAS

Crea una clase "ListaTareas" que contenga una lista de tareas pendientes. Implementa métodos para agregar una tarea, marcar una tarea como completada y mostrar todas las tareas

TIENDA ONLINE

Crea una clase "Producto" con atributos como nombre, precio y cantidad en stock. Luego, crea una clase "Tienda" que contenga una lista de productos disponibles y métodos para agregar productos, mostrar el inventario y realizar una compra.

PILA (STACK) BÁSICA

En programación, un "stack" es una estructura de datos que sigue el principio de LIFO (Last In, First Out), lo que significa que el último elemento agregado a la pila es el primero en ser retirado. Imagina una pila de platos: cuando apilas un nuevo plato, este se coloca en la parte superior de la pila, y cuando retiras un plato, siempre tomas el de arriba primero.

En Python, puedes implementar un stack utilizando una lista. Puedes agregar elementos a la pila utilizando el método `append()`, y puedes retirar elementos de la pila utilizando el método `pop()` sin ningún índice especificado, ya que `pop()` por defecto elimina y devuelve el último elemento de la lista.

Los stacks son útiles en muchas situaciones, como algoritmos de búsqueda y recorrido, manejo de llamadas a funciones (con la pila de llamadas), manejos de historial y navegación y más.

Crea una clase "Pila" que represente una pila (stack) básica. Implementa métodos para agregar elementos a la pila, quitar elementos y mostrar el contenido actual.

Por supuesto, estaré encantado de explicarte qué es un "stack" en el contexto de la programación y cómo se utiliza en Python.

SISTEMA DE GESTION DE BIBLIOTECA

Crea un sistema de gestión de una biblioteca utilizando clases en Python. Debes implementar las siguientes clases:

1. "Libro": Representa un libro con atributos como título, autor y número de ejemplares disponibles.
2. "Usuario": Representa a un usuario de la biblioteca con atributos como nombre, número de identificación y lista de libros prestados.
3. "Biblioteca": Representa la biblioteca en sí, con métodos para agregar libros, prestar libros a usuarios, devolver libros y mostrar el inventario.

EJERCICIOS PYTHON AVANZADO

HOJA 3C

TEMA: MANEJO DE CLASES Y OBJETOS (HERENCIA)

SISTEMA DE RESERVAS DE VUELOS

Imagina que estás desarrollando un sistema de reservas de vuelos para una aerolínea. Crea un sistema de clases que permita a los usuarios realizar reservas de vuelos. Aquí tienes una posible estructura:

- Clase base: `Vuelo`
 - Atributos: número de vuelo, origen, destino, capacidad máxima, lista de pasajeros
 - Métodos: agregar pasajero, verificar disponibilidad de asientos
- Clase derivada: `VueloEspecial` (hereda de `Vuelo`)
 - Atributos adicionales: motivo del vuelo especial (por ejemplo, vacaciones, trabajo)

Resuelve el problema creando instancias de estas clases y realizando reservas para diferentes vuelos y tipos de vuelos especiales.





SISTEMA DE GESTIÓN DE EMPLEADOS

Supongamos que estás construyendo un sistema de gestión de empleados para una empresa. Crea un sistema de clases que maneje la información de los empleados, incluyendo empleados a tiempo completo y empleados a tiempo parcial.

- Clase base: `Empleado`
 - Atributos: nombre, apellido, salario base
- Clase derivada: `EmpleadoTiempoCompleto` (hereda de `Empleado`)
 - Atributo adicional: bono anual
- Clase derivada: `EmpleadoTiempoParcial` (hereda de `Empleado`)
 - Atributo adicional: horas trabajadas por semana

Resuelve el problema creando instancias de estas clases y calculando los salarios totales para diferentes tipos de empleados.

Python avanzado - ejercicios

 Autoría	 Ana Raquel Martinez Carballo
 Etiquetas	
 Hora de creación	@28 de enero de 2024 23:17

Recursividad:

Tienes a tu disposición un conjunto de discos numerados del 1 al N y tres torres etiquetadas como A, B y C. La torre A contiene inicialmente todos los discos apilados en orden descendente, desde el disco N en la parte inferior hasta el disco 1 en la parte superior.

Tu tarea es implementar una solución **recursiva** para mover todos los discos desde la torre A hasta la torre C, siguiendo las reglas clásicas de las Torres de Hanoi:

1. Puedes mover un disco a una torre adyacente.
2. Solo puedes mover un disco a la cima de otra pila si esa pila está vacía o si el disco superior es más grande que el disco que estás colocando.

Debes definir una función llamada `torres_de_hanoi(n, origen, destino, auxiliar)` que, dado el número total de discos `n` y las torres de origen, destino y auxiliar, imprima los pasos necesarios para lograr el movimiento de todos los discos desde la torre A hasta la torre C.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- N de discos - N de torres - Torres : origen, destino, auxiliar	Mover disco de la torre A a la torre D Mover disco de la torre A a la torre B ...

Ejercicio 2:

Problema de Gestión de Inventario:

Imagina que trabajas en una empresa de logística y tu tarea es desarrollar un sistema de gestión de inventario. El inventario está representado como una lista

de productos ordenados por sus códigos. Cada producto se describe como un diccionario con las claves `'codigo'` y `'cantidad'`.

Tu objetivo es implementar una función **recursiva** que realice una búsqueda binaria en este inventario y devuelva la cantidad disponible para un producto específico, dado su código.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- Inventario de productos (json,dic,etc) - Código de producto buscado	Cantidad disponible para el producto 307: 80

Ejercicio 3:

Problema de Resolución de Laberinto:

Imagina que eres parte de un equipo de desarrollo de IA que se encarga de crear un sistema para que un robot resuelva laberintos. El laberinto está representado por una matriz, donde ciertos valores indican caminos permitidos (0), paredes (1), y la salida (9). Tu tarea es implementar una función **recursiva** que encuentre la ruta más corta para que el robot salga del laberinto.

Toma en cuenta los siguientes puntos:

1. La matriz representa el laberinto, donde los valores son:
 - 0: Camino permitido.
 - 1: Pared, no se puede atravesar.
 - 9: Salida del laberinto.
2. Debes implementar la función `resolver_labirinto` que utiliza recursividad para encontrar la ruta más corta desde una posición inicial hasta la salida.
3. La función debe devolver una lista de coordenadas que representan la ruta desde la posición inicial hasta la salida.
4. Puedes usar una lista de movimientos posibles: arriba ((-1, 0)), abajo ((1, 0)), izquierda ((0, -1)), derecha ((0, 1)).

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
<ul style="list-style-type: none"> - Laberinto (matriz) - Índice de inicio (fila) - Índice de inicio (columna) 	Camino para salir del laberinto: (1,1),(1,2), (),()...

Funciones Lambda:

Ejercicio 4:

Problema de Organización de Datos Empresariales:

Imagina que trabajas en una empresa internacional con equipos distribuidos en diferentes países. Cada equipo tiene una lista de empleados, representados como diccionarios, con información sobre el nombre, la edad y el rendimiento en proyectos recientes.

Tu tarea es organizar una lista consolidada de todos los empleados de la empresa. La organización debe seguir ciertas reglas:

1. Los empleados se deben ordenar por el rendimiento en proyectos recientes de forma descendente.
2. Para aquellos con el mismo rendimiento, se deben ordenar por edad de forma ascendente. Además, deseas agrupar a los empleados por país para un análisis más efectivo. Utiliza funciones **lambda**.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- Registro de empleados (json,dic,etc)	Empleados agrupados

Ejercicio 5:

Problema de Análisis de Datos de Ventas:

Imagina que eres parte de una empresa de comercio electrónico y tienes información detallada sobre las ventas de productos. Cada venta se representa como un diccionario, que incluye el nombre del producto, la fecha de venta, el monto de la venta y la ubicación del comprador. Realiza un análisis avanzado de estas ventas.

1. Filtra las ventas realizadas en el último trimestre del año.
2. Selecciona solo las ventas de productos con un monto superior a \$500.
3. Agrupa las ventas por ubicación del comprador.
4. Calcula el promedio del monto de venta para cada ubicación.
5. Ordena las ubicaciones por el promedio del monto de venta de forma descendente. Utiliza funciones ***lambda***.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- Registro de ventas (json,dic,etc)	Ubicaciones por promedio, ej. : [Chile, Ecuador]

Ejercicio 6:

Problema de Transformación y Filtrado de Nombres:

Imagina que te encuentras desarrollando una herramienta de procesamiento de nombres para una aplicación de gestión de contactos. Tienes una lista de nombres en el formato "Apellido, Nombre", realiza las siguientes tareas:

1. Utiliza la función ***lambda*** para transformar una lista de nombres completos al nuevo formato.
2. Filtra la lista para incluir solo los nombres que contienen al menos dos vocales y tienen una longitud mayor a 10 caracteres.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- Lista de nombres, ej: ["Pérez, Juan", "López, María"]	Nombres filtrados, ej: ['María López', 'José García']

Decoradores:

Ejercicio 7:

Logger con Tiempo de Ejecución

Imagina que estás desarrollando un sistema complejo que incluye múltiples funciones críticas. Para asegurarte de que todo funcione correctamente y para realizar un seguimiento del tiempo de ejecución de estas funciones, decides implementar un **decorador** de registro (logger) con tiempo de ejecución.

El decorador debería realizar las siguientes acciones:

1. Antes de llamar a la función original (puedes incluir cualquier función), debe imprimir un mensaje indicando que la función está a punto de ejecutarse.
2. Después de que la función se haya ejecutado, debe imprimir un mensaje que incluya el tiempo que tardó la función en ejecutarse.
3. Si la función original arroja una excepción, el decorador debe manejarla e imprimir un mensaje adecuado, indicando que se ha producido una excepción.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
-	- Invocando a la función 'mi_funcion'... - La función 'mi_funcion' ha tardado 0.0012459754943847656 segundos en ejecutarse. - Resultados de la función: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]

Ejercicio 8:

Decorador de Control de Acceso:

Imagina que estás trabajando en el desarrollo de un sistema para una aplicación de gestión de documentos en un entorno empresarial. Deseas implementar un **decorador** llamado `verificar_acceso_entorno` que permita controlar el acceso a funciones según el entorno de ejecución.

El decorador debe realizar las siguientes acciones:

1. Antes de ejecutar la función, verificar si el entorno de ejecución es "producción".

2. Si el entorno es "producción", permitir la ejecución de la función y mostrar un mensaje indicando que el acceso fue permitido en el entorno de producción.
3. Si el entorno no es "producción", evitar la ejecución de la función y mostrar un mensaje indicando que el acceso está restringido a entornos de producción.

Luego, aplica este decorador a dos funciones, `subir_documento` y `eliminar_documento`. Intenta ejecutar estas funciones con diferentes entornos y observa el comportamiento del decorador.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- entorno : 'produccion' / 'desarrollo'	- Acceso permitido / rechazado :
- subir_documento("Documento1")	- Documento subido
-eliminar_documento("Documento1")	- Documento eliminado

Ejercicio 9:

Ejercicio: Verificación de Inicio de Sesión con Decorador

Estás desarrollando un sistema de autenticación para una aplicación web y deseas implementar un sistema de inicio de sesión que verifique si las credenciales proporcionadas por el usuario son válidas antes de permitir el acceso a ciertas funciones. Además, deseas que una vez que el usuario haya iniciado sesión correctamente, se le proporcione información personal.

Implementa lo siguiente:

1. Un registro de usuarios que contenga información adicional, como el nombre completo y el correo electrónico.
2. Un decorador llamado `verificar_inicio_sesion` que acepte el nombre de usuario y la contraseña como argumentos. Este decorador verificará si las credenciales proporcionadas son válidas comparándolas con el registro de usuarios. Si las credenciales son válidas, la función decorada se ejecutará y se le pasará como argumento la información personal del usuario.
3. Una función llamada `informacion_usuario` que imprima la información personal del usuario después de que haya iniciado sesión correctamente.

Implementa este sistema de inicio de sesión utilizando **decoradores**.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- informacion_usuario("usuario1", "contrasena123")	- Inicio de sesión exitoso para el usuario usuario1. * Información personal del usuario: * Nombre completo: Juan Pérez - Inicio de sesión fallido. Verifica tu nombre de usuario y contraseña.

Memoización y decoradores:

Ejercicio 10:

Problema de Optimización de Subarreglo:

Imagina que estás trabajando en un sistema de análisis de datos y te han proporcionado una lista de números enteros. Tu tarea es desarrollar una función llamada `max_subarray_sum` que encuentre y devuelva la suma máxima de un subarreglo contiguo en la lista.

Por ejemplo, considera la lista `[1, -2, 3, 10, -4, 7, 2, -5]`. El subarreglo contiguo con la suma máxima es `[3, 10, -4, 7, 2]`, y la suma de esos elementos es `18`. Por lo tanto, la función debería devolver `18` para esta lista.

Implementa la función `max_subarray_sum` y, además, aplica **memoización** para mejorar su eficiencia en el cálculo de subarreglos de suma máxima en listas previamente procesadas.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- arreglo: [1, -2, 3, 10, -4, 7, 2, -5]	- 18

Ejercicio 11:

Ejercicio de Memoización en Costos de Envío

Imagina que estás trabajando en un sistema de gestión de costos de envío para una empresa de logística. El sistema debe calcular el costo de envío para diferentes destinos, distancias y pesos de paquetes. Implementa una función llamada `calcular_costo_envio` que tome como entrada un destino, una distancia en kilómetros y un peso en kilogramos, y devuelva el costo total del envío.

Requerimientos:

1. El costo base de envío es de \$5.0.
2. El costo por kilómetro de distancia es de \$0.1.
3. El costo por kilogramo de peso es de \$0.2.

Implementa la función de manera eficiente utilizando memoización para evitar recalculer el costo para los mismos destinos, distancias y pesos.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- destino_1 = "Ciudad A" - distancia_1 = 150 - peso_1 = 2.5	- Costo de envío con memoización (Ciudad A): 20.5

Ejercicio 12:

Ejercicio de Memoización en Análisis de Texto

Imagina que estás trabajando en un sistema de análisis de texto que requiere calcular la frecuencia de ocurrencia de palabras en un conjunto de documentos. Implementa una función llamada `calcular_frecuencia_palabras` que tome como entrada un texto y devuelva un diccionario que muestre la frecuencia de cada palabra en el texto.

1. La función debe ser capaz de manejar textos y ser insensible a mayúsculas/minúsculas (por ejemplo, "Hola" y "hola" se consideran la misma palabra).
2. Se deben excluir las palabras comunes (artículos, preposiciones, etc.) que no aportan información relevante al análisis.
3. Utiliza memoización para evitar recalculer la frecuencia de palabras para el mismo texto.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- texto 1 - texto 2	- frecuencia de palabras, conteo por palabra.

Tienes a tu disposición un conjunto de discos numerados del 1 al N y tres torres etiquetadas como A, B y C. La torre A contiene inicialmente todos los discos apilados en orden descendente, desde el disco N en la parte inferior hasta el disco 1 en la parte superior.

Tu tarea es implementar una solución **recursiva** para mover todos los discos desde la torre A hasta la torre C, siguiendo las reglas clásicas de las Torres de Hanoi:

1. Puedes mover un disco a una torre adyacente.
2. Solo puedes mover un disco a la cima de otra pila si esa pila está vacía o si el disco superior es más grande que el disco que estás colocando.

Debes definir una función llamada `torres_de_hanoi(n, origen, destino, auxiliar)` que, dado el número total de discos `n` y las torres de origen, destino y auxiliar, imprima los pasos necesarios para lograr el movimiento de todos los discos desde la torre A hasta la torre C.