Junta de Andalucía
Consejería de Educación y Deporte

UD 4

Arrays y cadenas de caracteres

–

Módulo de Programación
1º DAW

Autor: Fran Gómez
2025/2026

Objetivos

- **Comprender** los conceptos fundamentales de **arrays** y cadenas de caracteres (**strings**), incluyendo su **declaración** e **inicialización**.
- Implementar correctamente arrays unidimensionales y **multidimensionales** para almacenar y gestionar datos.
- **Acceder, modificar y recorrer** los elementos de un array de manera eficiente utilizando índices y estructuras de control (bucles).

Objetivos

- **Identificar** la necesidad del uso de arrays
- **Manipular cadenas** de caracteres utilizando funciones y métodos específicos del lenguaje, como concatenación, extracción de subcadenas, búsqueda y comparación.
- **Aplicar** los conocimientos de arrays y cadenas en la creación de soluciones sencillas y prácticas (ej. procesamiento de listas, manejo de texto).

RA y CE implicados

RA6 Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.

a) Se han escrito programas que utilicen arrays
g) Se han utilizado expresiones regulares en la búsqueda de patrones en cadenas de texto.

¿Cómo lo vamos a evaluar?

- Portfolio
- Ejercicios
- Cuestionario
- Examen escrito
- Proyecto

Índice de contenidos

I. Introducción a las Estructuras de Datos

1. **¿Qué es una estructura de datos?**
2. Estructuras **estáticas** vs. **dinámicas**.

II. Arrays (Tablas o Arreglos)

1. **Concepto de Array:**
   - Definición: Colección de elementos del **mismo tipo**.
   - Características: Tamaño **fijo**, acceso por **índice**.
2. **Arrays Unidimensionales:**
   - Declaración e Inicialización.
   - **Acceso** y **modificación** de elementos (índices).
3. **Recorrido de Arrays:** Uso de bucles (for, for-each).
   - Cálculo de la **suma** y el **promedio**.
   - Búsqueda de **mínimo** y **máximo**.
4. **Arrays Multidimensionales:**
   - Concepto (Matrices).
   - Declaración, Inicialización y Acceso.
   - Recorrido con **bucles anidados**.

III. Cadenas de Caracteres (Strings)

1. **Concepto de Cadena:**
   - Definición: Array de **caracteres**.
   - Diferencias y similitudes con un array tradicional.
2. **Declaración e Inicialización de Cadenas.**
3. **Operaciones Básicas:**
   - **Longitud** de la cadena.
   - **Concatenación** (unión).
   - Acceso a un **carácter** específico.
4. **Manipulación de Cadenas (Métodos/Funciones):**
   - **Comparación** de cadenas (igualdad).
   - **Subcadenas** (extracción).
   - **Búsqueda** de caracteres o patrones.
   - Conversión a **mayúsculas/minúsculas**.
   - Conversión entre **cadenas y otros tipos de datos**.

IV. Aplicaciones y Ejercicios Prácticos

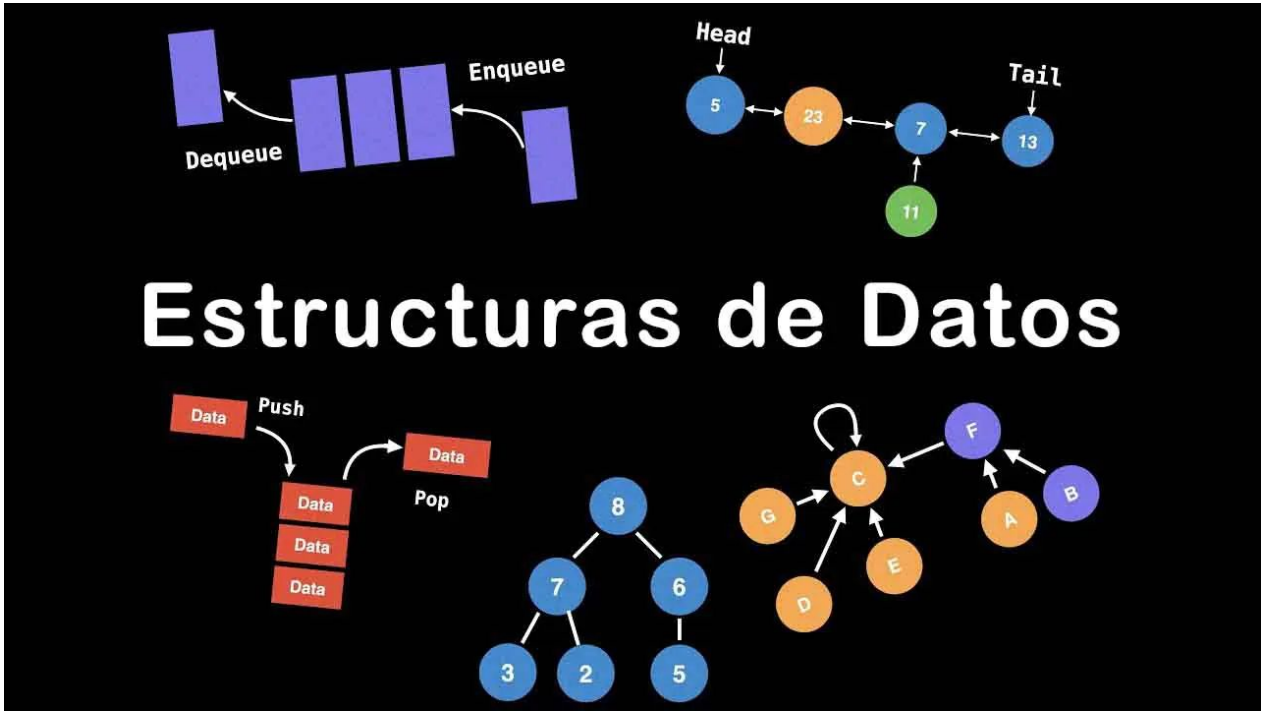
1. Uso de arrays para simular **listas** o **almacenamiento** de datos.
2. Uso de cadenas en el **procesamiento de texto** (ej. inversión de palabras, conteo de vocales).
3. Ejercicios que combinan ambos conceptos (**arrays de cadenas**).

Introducción

¿Cómo Almacenar y Gestionar Grandes Cantidades de Datos Similares?

Por ejemplo: Necesitas almacenar las notas de los 20 alumnos del curso.

```
1  nota1 = 5
2  nota2 = 7
3  nota3 = 4
4  ...
5  nota20 = 6
```

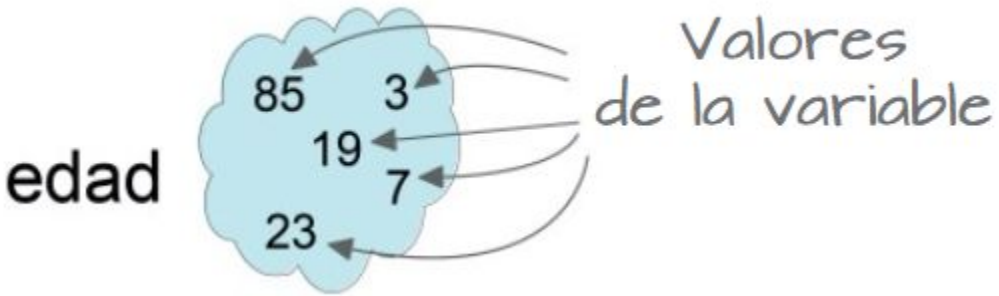


Introducción: Estructuras de datos

¿Cuántos valores puede almacenar simultáneamente una variable?

```
edad = 6;
edad = 23;
```

Variables escalares



Tablas

Introducción: Estructuras de datos

Vamos a clasificar las primeras estructuras de datos que estudiaremos en:

- Arrays (Estáticas): Tamaño fijo, elementos homogéneos (del mismo tipo: enteros, strings, etc.).
- Listas (Dinámicas): Tamaño variable (siguiente tema).

Your Wooclap question will appear here

1. Install the **Chrome** or **Firefox** extension 🧩
2. Click on “Add a Wooclap vote” and make sure you are logged into your Wooclap account 👨‍💻
3. Ensure you are in **presentation mode** on Google Slides 🖥️

wooclap

Your Wooclap question will appear here

1. Install the **Chrome** or **Firefox** extension 🧩
2. Click on “Add a Wooclap vote” and make sure you are logged into your Wooclap account 👨‍💻
3. Ensure you are in **presentation mode** on Google Slides 🖥️

wooclap

Tablas o Arrays: Concepto

Definición: estructura de datos (de tipo objeto) que almacena un conjunto de elementos de un mismo tipo, accedidos mediante un índice.

Índice de un array: número natural, comenzando en 0, que identifica las sucesivas posiciones de un array. El último índice sería N-1, donde N es el número de elementos del array.

El Array en Python: La Lista

Definición:

- En Python no existen los arrays tradicionales como en otros lenguajes, pero se implementan usando listas.
- Colección de elementos
- Homogéneos o heterogéneos: aunque nos enfocaremos en homogéneos.
- Ordenada según se van insertando los elementos en la lista
- Mutable: su contenido puede modificarse (ya lo veremos en próximos temas)

Declaración e Inicialización

- **Sintaxis simple:**

```
mi_array = [10, 20, 30, 40]
```

- **Lista vacía:**

```
otra_lista = []
```

- **Inicialización rápida (Pre-asignación):**

```
ceros = [0] * 5
```

Acceso y Modificación por Índice

El concepto clave del **índice** (0 a N - 1).

Acceso:

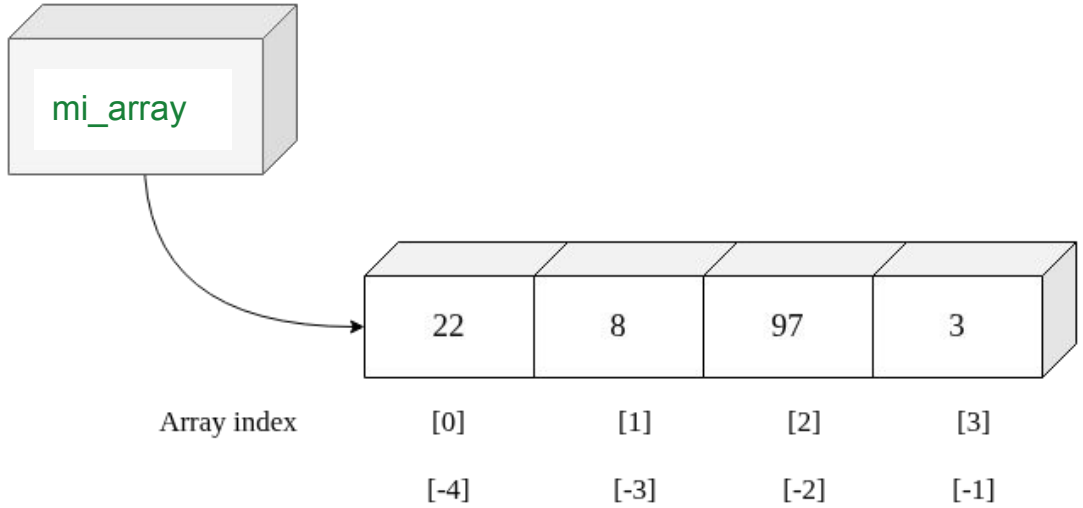
mi_array[0] (Primer elemento)

Modificación:

mi_array[2] = 97

Acceso Negativo:

mi_array[-1] (Último elemento)



Recorrido: Bucles y Listas

- **Recorrido tradicional (por índice):**

```
for i in range(len(mi_array)):
    print(mi_array[i])
```

- **Recorrido for-each (por elemento):**

```
for elemento in mi_array:
    print(elemento)
```

Métodos Básicos de Listas

- `append()`: Añade un elemento al final de la lista

  list.append(5)
- `insert()`: Añadir en posición

  list.insert(3, 'y')
- `len()`: Devuelve la longitud

  len(list)
- `remove()`: Elimina la primera ocurrencia de un elemento de la lista

  list.remove('p')
- `pop()`: Elimina un elemento y lo devuelve de la posición marcada o del final

  list.pop(4)

Python Arrays

Ejercicio 2: Suma y Recorrido

Ejercicio de Codificación: Crea un array de 5 números y usa los dos tipos de bucles para calcular y mostrar la suma de sus valores.

0	1	2	3	4
25	12	30	10	8

Ejercicio 3 (Debugging): Errores de Índice

Explica qué está pasando aquí y cómo corregirlo.

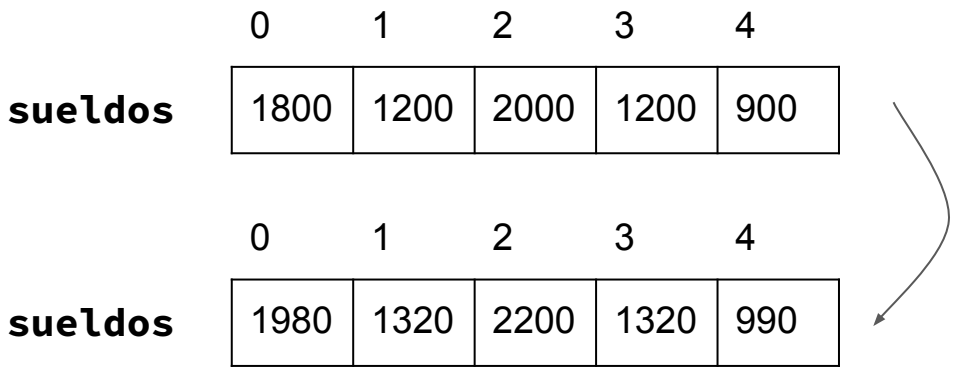
```
1  a = [1,2,3,4]
2  for i in range(5):
3      print(a[i])
```

```
Shell
2
3
4
Traceback (most recent call last):
  File "<string>", line 3, in <module>
IndexError: list index out of range

>>>
```

Ejercicio 4: Recorrer un array

Ejercicio de codificación en Python

- Crea una tabla como la de la imagen, que representa los sueldos de los empleados de una empresa.
- Aumentarles el sueldo un 10% a cada empleado de forma que el array de sueldos quede actualizado



Ejercicios

Ejercicio 5

Diseñar un programa que solicite al usuario que introduzca por teclado 5 números decimales. A continuación, mostrar los números en el mismo orden que se han introducido.

Ejercicio 6

Escribir una aplicación que solicite al usuario cuántos números desea introducir. A continuación, introducir por teclado esa cantidad de números enteros, y por último, mostrar en el orden inverso al introducido. Hacerlo con While y con for range.

Ejercicio 7

Diseñar la función: def maximo (t), que devuelva el máximo valor contenido en la tabla t.

Operaciones con arrays

Básicas

- Recorrido (ya lo hemos visto)
- Impresión (ya lo hemos visto)
- Acceso y modificación (ya lo hemos visto)
- Inicialización masiva (ya lo hemos visto)
- División
- Adición
- Inserción
- Unión
- Eliminación
- Extracción

Avanzadas

- Búsqueda
- Ordenación
- Comparación
- Copia
- Intercambio

División o Slice. Operador :

lista [inicio : fin : paso]

```
1  my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
2  print("my_list =", my_list)
3
4  # get a list with items from index 2 to index 4 (index 5 is not included)
5  print("my_list[2: 5] =", my_list[2: 5])
6
7  # get a list with items from index 2 to index -3 (index -2 is not included)
8  print("my_list[2: -2] =", my_list[2: -2])
9
10 # get a list with items from index 0 to index 2 (index 3 is not included)
11 print("my_list[0: 3] =", my_list[0: 3])
12
13
```

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
my_list[2: 5] = ['o', 'g', 'r']
my_list[2: -2] = ['o', 'g', 'r']
my_list[0: 3] = ['p', 'r', 'o']
```

División. Operador :

Ejercicio 8

Crea un array con tu nombre y apellido y divídelo en dos arrays, uno para el nombre y otro para el apellido.

Muestra tanto el array inicial como los dos arrays del resultado.

Adición. Método append()


```
fruits = ['apple', 'banana', 'orange']
print('Original List:', fruits)

fruits.append('cherry')

print('Updated List:', fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'orange', 'cherry']
```

Añadir. Método append()

Ejercicio 9

Crea un array con los nombres de varios alumnos y pide el nombre de un nuevo alumno al usuario para añadirlo al final de la lista.

Muestra tanto el array inicial y el resultado.

Inserción. Método insert()


```
fruits = ['apple', 'banana', 'orange']
print("Original List:", fruits)

fruits.insert(2, 'cherry')

print("Updated List:", fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'cherry', 'orange']
```

Inserción. Método insert()

Ejercicio 10

Partimos del array de alumnos del ejercicio anterior.

Imagina que quieres añadir un nuevo alumno pero su sitio está entre el 2º y el 3º en lugar de al final.

Imprime original y resultado

Unir listas. Método extend()


```
numbers = [1, 3, 5]
print('Numbers:', numbers)

even_numbers  = [2, 4, 6]
print('Even numbers:', numbers)

# adding elements of one list to another
numbers.extend(even_numbers)

print('Updated Numbers:', numbers)
```

Output

```
Numbers: [1, 3, 5]
Even numbers: [2, 4, 6]
Updated Numbers: [1, 3, 5, 2, 4, 6]
```

Unir listas. Método extend()

Ejercicio 11

Ahora imagina que tenemos dos aulas con tres alumnos cada una y queremos unir ambas aulas en una sóla.

Imprime el nuevo array resultante (no modifiques los arrays originales)

Eliminación. Método remove()


```
lista.remove(valor)
```

Parte	Descripción
`lista`	La lista sobre la que se ejecuta la acción.
`valor`	El elemento **exacto** que deseas eliminar de la lista.
Retorno	**No devuelve ningún valor** (`None`). Modifica la lista *in situ*.

```
numbers = [2,4,7,9]

# remove 4 from the list
numbers.remove(4)

print(numbers)
```

Output

```
[2, 7, 9]
```

Eliminación. Método remove()

Ejercicio 12

Dada la siguiente lista de frutas, usa el método `remove()` para:

1. Eliminar la fruta **"Manzana"** de la lista.
2. Intentar eliminar la fruta **"Pera"**.

Lista Inicial: `frutas = ["Plátano", "Manzana", "Cereza", "Naranja", "Manzana", "Kiwi"]`

Ejercicio 12 (antiguo)

Crea una lista con los 10 primeros números.

Elimina los números pares de la lista anterior.

Imprime lista original y resultante.

Extracción. Método pop()


```
elemento_eliminado = lista.pop(indice)
```

Parte	Descripción
`lista`	La lista sobre la que se ejecuta la acción.
`indice`	La posición **opcional** del elemento a eliminar (empezando en 0).
Retorno	**Devuelve el elemento** que fue eliminado de la lista.

```
1  vocales = ['a','e','i','o','u']
2  vocales.pop(2)
3  print(vocales)
```

```
['a', 'e', 'o', 'u']
```

Extracción. Método pop()

Ejercicio 13

Dada la siguiente lista de animales, usa el método `pop()` para:

1. Eliminar y guardar el último animal de la lista.
2. Eliminar y guardar el animal que se encuentra en el índice `1`.

Lista Inicial: `animales = ["Gato", "Perro", "Conejo", "Loro", "Pez"]`

Ejercicio 13 (antiguo)

Crea una lista con los 10 primeros números.

Extrae los números primos de la lista anterior y añadelos a una nueva lista.

Imprime lista original al principio y al final así como la lista resultante.

Más métodos…

Existen más métodos y funciones útiles para los arrays en Python; podemos consultarlos en su documentación:

https://docs.python.org/es/3/tutorial/datastructures.html

Ejercicios. Ejercicio 14

Gestión de Inventario 📦

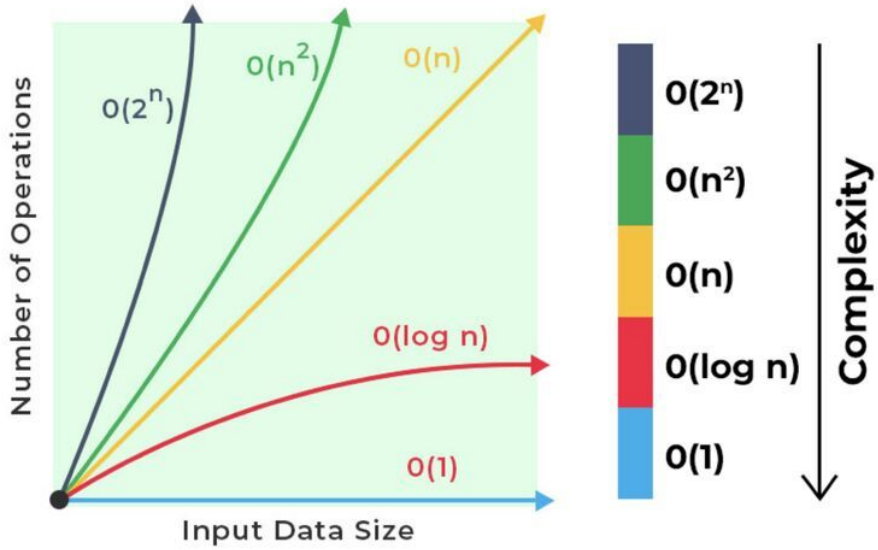
Tu objetivo es simular las operaciones diarias en el inventario de una pequeña tienda utilizando una **Lista** de Python.

Inventario Inicial: inventario = ["Manzanas", "Plátanos", "Naranjas", "Peras"]

1. **Llegada de Nueva Mercancía:**
   - Llega un envío pequeño de **"Uvas"** y **"Kiwi"**. **Agrégalo** al final del inventario (**Adición**).
   - Llega una caja grande de **"Melones"** que debe ser tratada como un conjunto separado inicialmente y luego **unida** al inventario principal (**Unión**).
2. **Reordenamiento:**
   - Por un error de etiquetado, los **"Plátanos"** (que están en el índice 1) deben ir justo después de las **"Manzanas"** (índice 0). **Inserta** la etiqueta **"Plátanos Frescos"** en el índice 1, desplazando los demás.
3. **Venta y Descarte:**
   - Se **vende** el último producto añadido (**"Melones"**). **Extrae** (usando `pop`) este producto y guárdalo en una variable llamada `producto_vendido`.
   - Se descubre que las **"Peras"** (que ahora están en una nueva posición debido a las inserciones/adiciones) están magulladas y deben ser **eliminadas** del inventario sin guardarlas.
4. **Reporte de Emergencia:**
   - Crea un nuevo array llamado `reporte_urgente` que contenga solo los primeros 4 elementos restantes del inventario para un reporte rápido (**División/Slice**).
5. **Recorrido Final:**
   - Usa un bucle `for` para **imprimir** el `reporte_urgente`, mostrando siempre el índice seguido del producto (Ej: `Índice 0: Manzanas`).

Operaciones Avanzadas

Aquí vamos a hablar de algoritmos para resolver problemas típicos pero más complejos que los que resuelven las operaciones básicas.



Buscar un elemento en un array

Problema:

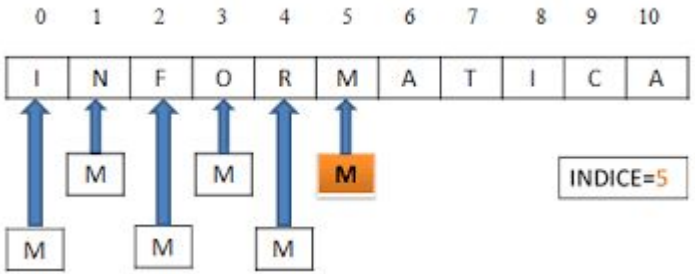
Encontrar si un elemento específico existe en una lista y, si es así, dónde está (su índice).

Algoritmos de búsqueda

1. Búsqueda secuencial → aplica a arrays desordenados

2. Búsqueda binaria → aplica a arrays ordenados

Búsqueda secuencial o lineal




```java
public static int busquedaLineal(int[] array, int clave) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == clave) {
            return i; // Retorna la posición del elemento.
        }
    }
    return -1; // No encontrado.
}
```

Recorro todo el array en busca del elemento, preguntando uno a uno.

complejidad O(n)

Búsqueda en Listas/Arrays en Python

En Python tenemos el operador **in** y el método **index** que nos facilitan el trabajo.

```
participantes = ["Ana", "Luis", "Marta", "Pedro", "Julia"]

# Verificar si "Carlos" está en la lista
if "Carlos" in participantes:
    print("Carlos está participando.")
else:
    print("Carlos NO está en la lista de participantes.")

# Encontrar la posición de "Marta"
try:
    posicion_marta = participantes.index("Marta")
    print(f"Marta se encuentra en la posición: {posicion_marta}")
except ValueError:
    print("Marta no fue encontrada.")
```

Buscar un elemento en un array

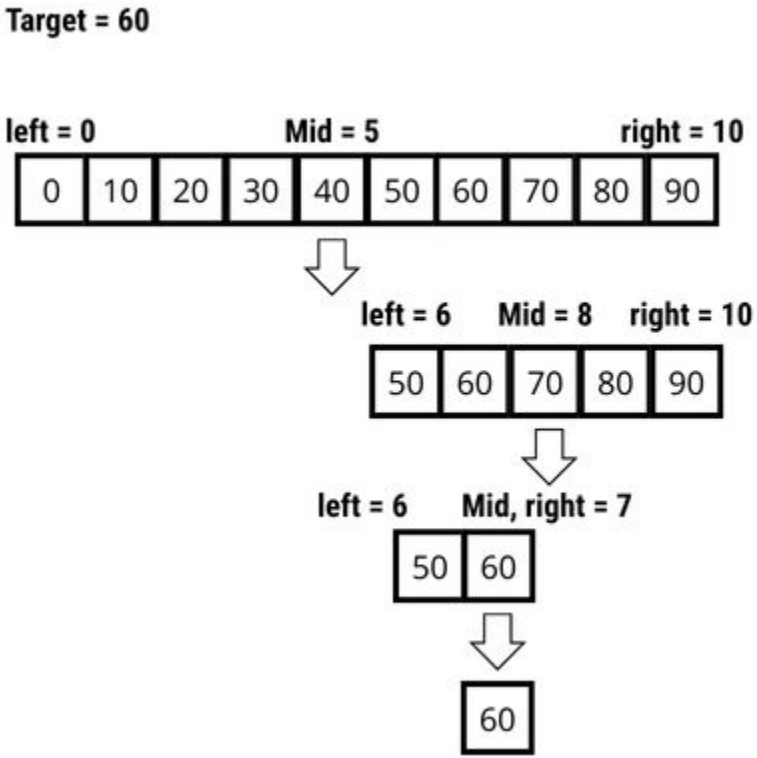
Ejercicio 15:

1. Crea un método que reciba un número y un array y devuelva la posición donde ha encontrado el número o -1 si no ha encontrado el número.

2. Crea un array con 5 números aleatorios y busca dentro de él un número introducido por el usuario, invocando el método anterior.

3. Compara el resultado de tu método con el del método index

Búsqueda binaria



Ejercicio 16

```python
def busqueda_binaria(lista_ordenada, objetivo):
…
# Ejemplo de uso
numeros = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
buscar = 23

indice = busqueda_binaria(numeros, buscar)

if indice != -1:
    print(f"El número {buscar} se encuentra en el índice: {indice}")
else:
    print(f"El número {buscar} no se encuentra en la lista.")
```

complejidad O(logn)

Ordenación de arrays

En Python puedes ordenar de dos formas:

- Método .sort() → modifica la lista original
- Función sorted() → crea una nueva lista

Ordenación de arrays: método .sort()

Sintaxis:

```
list.sort(reverse=True|False, key=myFunc)
```

```
numeros = [4, 1, 7, 3, -1]
numeros.sort()
print(numeros)
# Salida: [-1, 1, 3, 4, 7]

palabras = ["gato", "perro", "ave"]
palabras.sort()
print(palabras)
# Salida: ['ave', 'gato', 'perro']
```

Ordenación de arrays: método sorted()

Sintaxis:

```
sorted(iterable, key=key, reverse=reverse)
```

```python
numeros = [4, 1, 7, 3, -1]
numeros_ordenados = sorted(numeros)

print(numeros_ordenados)
# Salida: [-1, 1, 3, 4, 7]

print(numeros)
# Salida: [4, 1, 7, 3, -1] (La original no cambió)
```

Ordenación de arrays: método sorted()

Ejercicio 17

Necesitas imprimir un informe rápido de todas las tareas ordenadas alfabéticamente, pero sin que las mayúsculas afecten el orden (es decir, "comprar café" debe ir antes que "Llamar al cliente").

tareas_dia = ["Revisar Email", "Llamar al cliente", "comprar café", "Actualizar reporte", "Planificar reunión"]

Requisito crucial: Esta es solo una vista temporal. La lista original `tareas_dia` debe **permanecer en su orden original** después de que imprimas este informe.

Tu Tarea:

1. Usa la función `sorted()` para crear una **nueva** lista llamada `informe_ordenado`.
2. Utiliza el argumento `key=str.lower` para que el orden ignore mayúsculas/minúsculas.
3. Imprime la nueva lista `informe_ordenado`.
4. Vuelve a imprimir la lista original `tareas_dia` para demostrar que no ha cambiado.

Ordenación de arrays: método .sort()

Después de la reunión, decides que la lista `tareas_dia` debe reorganizarse permanentemente. Quieres que las tareas más largas (quizás las más complejas) aparezcan al final de la lista.

Requisito crucial: Debes modificar la lista `tareas_dia` **en su lugar** (in-place) para que refleje este nuevo orden.

Tu Tarea:

1. Usa el método `.sort()` directamente sobre la lista `tareas_dia`.
2. Utiliza el argumento `key=len` para ordenar las tareas por su longitud (de más corta a más larga).
3. Imprime la lista `tareas_dia` para mostrar cómo ha sido modificada permanentemente.

Ordenación de arrays

Ejercicio 18:

Crea un array con elementos desordenados.

Invoca primero la función de búsqueda secuencial y luego la binaria.

Comparar resultados entre la búsqueda binaria y la secuencial.

Después, ordena el array, vuelve a ejecutar la búsqueda secuencial y binaria y ahora compara los resultados con los del array desordenado.

Ordenación de arrays

Algoritmos de ordenación

1. Ordenación burbuja
2. Ordenación por selección
3. Ordenación por inserción

Comparación de arrays

La comparación de arrays se puede hacer para verificar si son idénticos, lo que significa:

- Que tienen la misma longitud
- Que tienen los mismos elementos en el mismo orden.

Esto se logra a menudo mediante métodos específicos del lenguaje de programación, como el operador `==` en Python o `Arrays.equals()` en Java, que devuelve `true` si ambos arrays son iguales.

```python
lista_a = [10, 20, 30]
lista_b = [10, 20, 30]
lista_c = [30, 20, 10] # Mismos elementos, orden diferente
lista_d = [10, 20]     # Longitud diferente

print(f"A == B: {lista_a == lista_b}") # True
print(f"A == C: {lista_a == lista_c}") # False (el orden importa)
print(f"A == D: {lista_a == lista_d}") # False (la longitud importa)
```

Comparación de arrays

Ejercicio 19

Crea un algoritmo que compare dos arrays. Sigue estos pasos:

1. Crea el pseudocódigo
2. Impleméntalo en Python
3. Prueba los tres casos de uso:
    a. Son iguales
    b. Tienen diferente longitud
    c. Tienen diferentes valores
    d. Tienen los mismos valores pero diferente orden

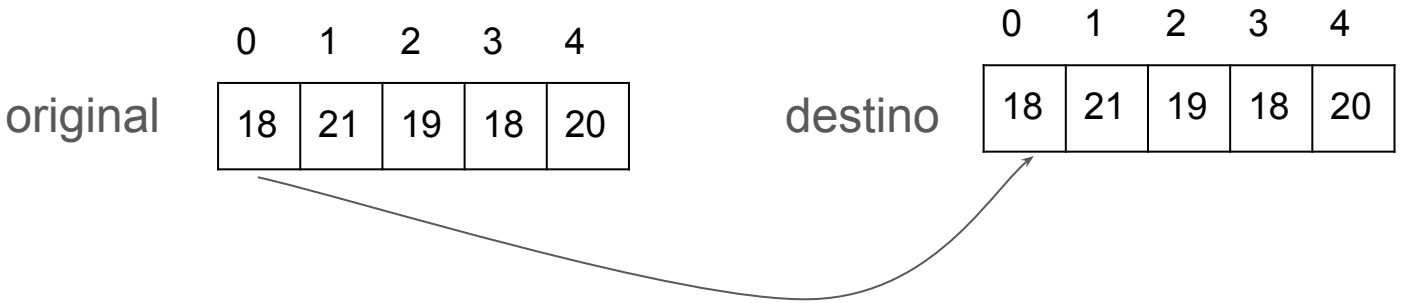
Ampliación: pasa el algoritmo a una función que retorne un valor booleano

Copiar arrays

El procedimiento para realizar manualmente la copia exacta de una tabla consiste en:

1. Crear una nueva tabla, que llamaremos *destino* o *copia*, del mismo tipo y longitud que la tabla original.

2. Recorrer la tabla original, copiando el valor de cada elemento en su lugar correspondiente en la tabla destino.



Copiar arrays

Forma **errónea** de hacerlo:

- Asignación → No se copia sino que se le da otro nombre más a la lista

```
# Lista original
lista_a = [1, 2, 3]

# 'Copia' por asignación
lista_b = lista_a

# Modificamos lista_b
lista_b.append(4)

print(f"Lista B: {lista_b}") # Salida: [1, 2, 3, 4]
print(f"Lista A: {lista_a}") # Salida: [1, 2, 3, 4]  <-- ¡La original cambió!
```

Copiar arrays

Formas de hacerlo correctamente:

- **Método .copy()**: nueva_lista = original.copy()
- **Slicing (rebanado)**: nueva_lista = original[:]
- **Manualmente**: como vimos al principio

```
lista_a = [1, 2, 3]

# Copia superficial
lista_b = lista_a.copy() # O lista_a[:]

# Modificamos lista_b
lista_b.append(4)

print(f"Lista B: {lista_b}") # Salida: [1, 2, 3, 4]
print(f"Lista A: {lista_a}") # Salida: [1, 2, 3] <-- ¡La original NO cambió!
```

Copiar arrays

Ejercicio 20

1. Crea un array llamado *original* con los números del 1 al 5.
2. Crea una copia llamada *copia1* utilizando el método copy
3. Crea otra copia llamada *copia2* usando slicing
4. Crea otra copia de manera manual llamada *copia3*
5. Asigna el array original a una variable llamada copia4. ¿Qué pasa si eliminas el último elemento de copia4? ¿y si eliminas el primer elemento de copia3?
6. Compara los 4 arrays usando el algoritmo del ejercicio 19

Arrays de Dos Dimensiones (Matrices) en Python

Un array de dos dimensiones se puede visualizar como una **tabla** o una **cuadrícula**, donde los datos se organizan en **filas** y **columnas**. En Python, esto se consigue anidando listas, donde cada lista interior representa una fila.

1. Creación de una Matriz

Definimos una matriz simple de 3 x 3 (3 filas y 3 columnas):

Python

```python
# La lista exterior contiene las filas.
# Cada lista interior es una fila.
matriz = [
    [1, 2, 3],  # Fila 0
    [4, 5, 6],  # Fila 1
    [7, 8, 9]   # Fila 2
]

print(matriz)
```

Matrices en Python

2. Acceso a Elementos

Para acceder a un elemento, necesitas dos índices: **primero la fila** y **luego la columna**.

matriz[fila][columna]

```
# Ejemplo de acceso:
elemento = matriz[1][2]
print(f"El elemento en la Fila 1, Columna 2 es: {elemento}")

# Ejemplo de modificación:
matriz[2][0] = 10
print(f"Matriz después de la modificación: {matriz}")
```

```
matriz = [
    [1, 2, 3],  # Fila 0
    [4, 5, 6],  # Fila 1
    [7, 8, 9]   # Fila 2
]
```

Operación	Código Python	Descripción	Resultado
Acceder a un elemento	matriz[1][2]	Fila 1 (la segunda fila), Columna 2 (el tercer elemento).	6
Acceder a una fila completa	matriz[0]	Toda la Fila 0.	[1, 2, 3]
Modificar un elemento	matriz[2][0] = 10	Cambia el primer elemento de la Fila 2 (el 7) por 10.	10

Matrices en Python

3. Recorrido de una Matriz (Iteración)

Para procesar todos los elementos de la matriz, normalmente se usan **dos bucles `for` anidados**:
el bucle exterior recorre las filas, y el bucle interior recorre las columnas de esa fila.

Python

```python
print("\nRecorriendo la matriz:")
for fila in matriz:
    for elemento in fila:
        print(elemento, end=" ") # Imprime el elemento y un espacio, sin salto de línea
    print() # Salto de línea después de terminar una fila

# Salida:
# 1 2 3
# 4 5 6
# 10 8 9
```

```python
matriz = [
    [1, 2, 3],  # Fila 0
    [4, 5, 6],  # Fila 1
    [7, 8, 9]   # Fila 2
]
```

Matrices en Python

Ejercicio 21

Dada la siguiente matriz que representa un tablero de juego:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

1. Escribe el código en Python para declarar e inicializar esta matriz como una lista anidada.
2. Accede e imprime el valor que se encuentra en la posición:
   - Fila 1, Columna 2 (Recuerda: la indexación empieza en 0) (el número 6).
   - El elemento central de la matriz (el número 5).

Ejercicio 21 (continuación)

Usando la matriz M anterior:

3. Escribe un bucle que recorra **solo la segunda fila** (índice 1) e imprima cada uno de sus elementos.

4. Escribe un bucle anidado que recorra **toda la matriz** e imprima todos sus elementos, separados por espacios en la misma línea para simular la matriz visualmente.

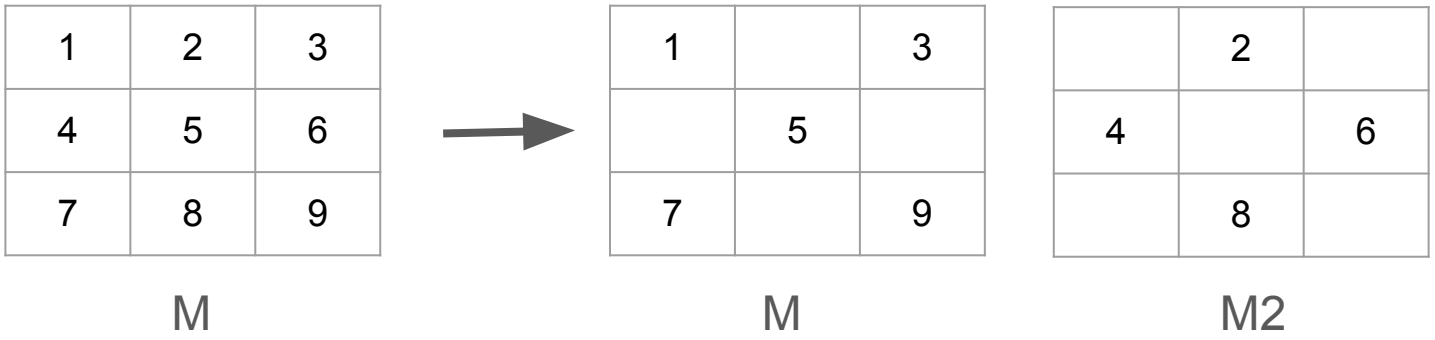
$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrices en Python

Ejercicio 21 (continuación)

Usando la matriz M anterior:

5. Recorre la matriz y ve extrayendo los números múltiplos de 2 a otra matriz en su misma posición, de modo que al final quede así:



Ejercicio 22

Recorrer una matriz (array bidimensional)

1. Inicializa un array de dos dimensiones llamado *matriz* con el contenido y forma siguiente:

A	B	C
D	E	F
G	H	I

Inténtalo primero con un while y luego con un for-each

2. Imprime *matriz* para que se muestre por consola lo siguiente:

A D G B E H C F I

Ejercicio 23

Dada la matriz

$$P = \begin{bmatrix} 5.0 & 10.5 \\ 8.2 & 15.0 \\ 1.5 & 3.0 \end{bmatrix}$$

Donde la Columna 0 son Precios de Mayorista y la Columna 1 son Precios de Minorista.

1. Crea la matriz P en Python.
2. Escribe un algoritmo que calcule e imprima la **suma total de los Precios de Minorista** (Columna 1).

Matrices en Python

Ejercicio 24

Dada una matriz de números enteros cualquiera:

1. Escribe un algoritmo que recorra todos los elementos de la matriz y **cuente cuántos números primos** contiene.

2. Imprime el conteo final.

Cadenas de caracteres: Introducción

- **¿Qué es una cadena?**
  - Una cadena es una **secuencia inmutable** de caracteres (letras, números, símbolos).
  - Sirven para almacenar texto.
- **Creación de Cadenas:**
  - Las cadenas se definen utilizando comillas simples (`'`) o dobles (`"`). Ambas son intercambiables, pero deben coincidir.

Ejemplos:

```
saludo_simple = 'Hola mundo'
saludo_doble = "¡Hola Python!"
```

Cadenas Multilínea: Usando tres comillas (`"""` o `'''`).

```
parrafo = """
Esto es un texto
que ocupa varias líneas
"""
```

- **Inmutabilidad (Concepto Clave):** Una vez que se crea una cadena, no se puede cambiar directamente un carácter individual. Cualquier "modificación" crea una nueva cadena.

Indexación

Las cadenas son secuencias, lo que significa que podemos acceder a sus elementos por posición.

- **Indexación (Acceso a un Carácter):**
  - Python utiliza **índices basados en cero** (el primer carácter es el índice `0`).
  - **Indexación Negativa:** Permite acceder a los caracteres desde el final. El último carácter es el índice `-1`.

Carácter	P	y	t	h	o	n
Índice positivo	0	1	2	3	4	5
Índice negativo	-6	-5	-4	-3	-2	-1

```
cadena = "Python"
print(cadena[0])  # Salida: 'P'
print(cadena[-1]) # Salida: 'n'
```

Slicing

- Permite extraer una subcadena (una porción de la cadena original).
- **Sintaxis:** `cadena[inicio:fin:paso]`
  - **`inicio`** es inclusivo (se incluye).
  - **`fin`** es exclusivo (no se incluye, se detiene antes).
  - **`paso`** (opcional) indica el salto.
- **Ejemplos:**

```
lenguaje = "Programacion"
print(lenguaje[0:5])   # Salida: 'Progr'
print(lenguaje[5:])    # Salida: 'amacion' (hasta el final)
print(lenguaje[:4])    # Salida: 'Prog' (desde el inicio)
print(lenguaje[::2])   # Salida: 'Pormco' (cada dos caracteres)
print(lenguaje[::-1])  # Salida: 'noicamar' (invertir la cadena)
```

Operaciones Básicas

Concatenación: Unir dos o más cadenas con el operador `+`.

```
nombre = "Juan"
apellido = "Perez"
nombre_completo = nombre + " " + apellido
# Salida: 'Juan Perez'
```

Multiplicación/Repetición: Repetir una cadena con el operador `*`.

```
print("-" * 20) # Salida: '--------------------'
```

Longitud: Usar la función `len()` para saber cuántos caracteres tiene una cadena.

```
texto = "Clase"
print(len(texto)) # Salida: 5
```

Métodos

Los métodos son funciones específicas que se aplican directamente a un objeto *string*.

Método	Descripción	Ejemplo
`.lower()`	Convierte toda la cadena a minúsculas.	`'HOLA'.lower()` → `'hola'`
`.upper()`	Convierte toda la cadena a mayúsculas.	`'hola'.upper()` → `'HOLA'`
`.capitalize()`	La primera letra mayúscula y el resto minúsculas.	`'python'.capitalize()` → `'Python'`
`.strip()`	Elimina espacios en blanco al inicio y al final.	`' hola '.strip()` → `'hola'`
`.replace(a, b)`	Reemplaza todas las ocurrencias de `a` por `b`.	`'abc'.replace('b', 'x')` → `'axc'`
`.split(separador)`	Divide la cadena en una **lista** de subcadenas usando un separador.	`'a,b,c'.split(',')` → `['a', 'b', 'c']`
`.join(iterable)`	Une los elementos de un iterable (como una lista) usando la cadena como separador.	`'-'.join(['a', 'b'])` → `'a-b'`
`.find(sub)`	Devuelve el índice de la primera ocurrencia de `sub`. Si no la encuentra, devuelve `-1`.	`'texto'.find('e')` → `1`
`.startswith(sub)`	Devuelve `True` si la cadena comienza con `sub`.	`'Python'.startswith('P')` → `True`

Formato de Cadenas

En la programación moderna de Python, la forma más recomendada para construir cadenas complejas con variables es usando las **f-strings** (cadenas formateadas).

- **F-Strings (Recomendado):** Se usa una `f` antes de la primera comilla. Las variables se insertan entre llaves `{}`.

```
producto = "Laptop"
precio = 850.50

mensaje = f"El {producto} cuesta ${precio:.2f}."
# Salida: El Laptop cuesta $850.50. (El .2f es para 2 decimales)
```

Ejercicio 25

1. **Inversor de Palabras:**
   - Pide al usuario una palabra (ej: `"radar"`).
   - Imprime la palabra invertida (debe usar *slicing* `[::-1]`).
   - *Desafío extra:* Comprobar si es un palíndromo (si la palabra original es igual a la invertida).
2. **Limpieza de Datos:**
   - Tienes la siguiente cadena: `" juan.perez@dominio.com "`.
   - **Tarea A:** Elimina los espacios en blanco del principio y el final.
   - **Tarea B:** Conviértela a mayúsculas.
   - **Tarea C:** Separa el nombre del dominio (`juan.perez` y `dominio.com`) usando el método `.split()` y el separador `@`.
3. **Etiqueta HTML:**
   - Usa f-strings para crear una etiqueta HTML que contenga una variable.
   - Variables: `clase = "titulo"` y `contenido = "Mi Encabezado"`.
   - Resultado deseado: `<h1 class="titulo">Mi Encabezado</h1>`

Ejercicio 26

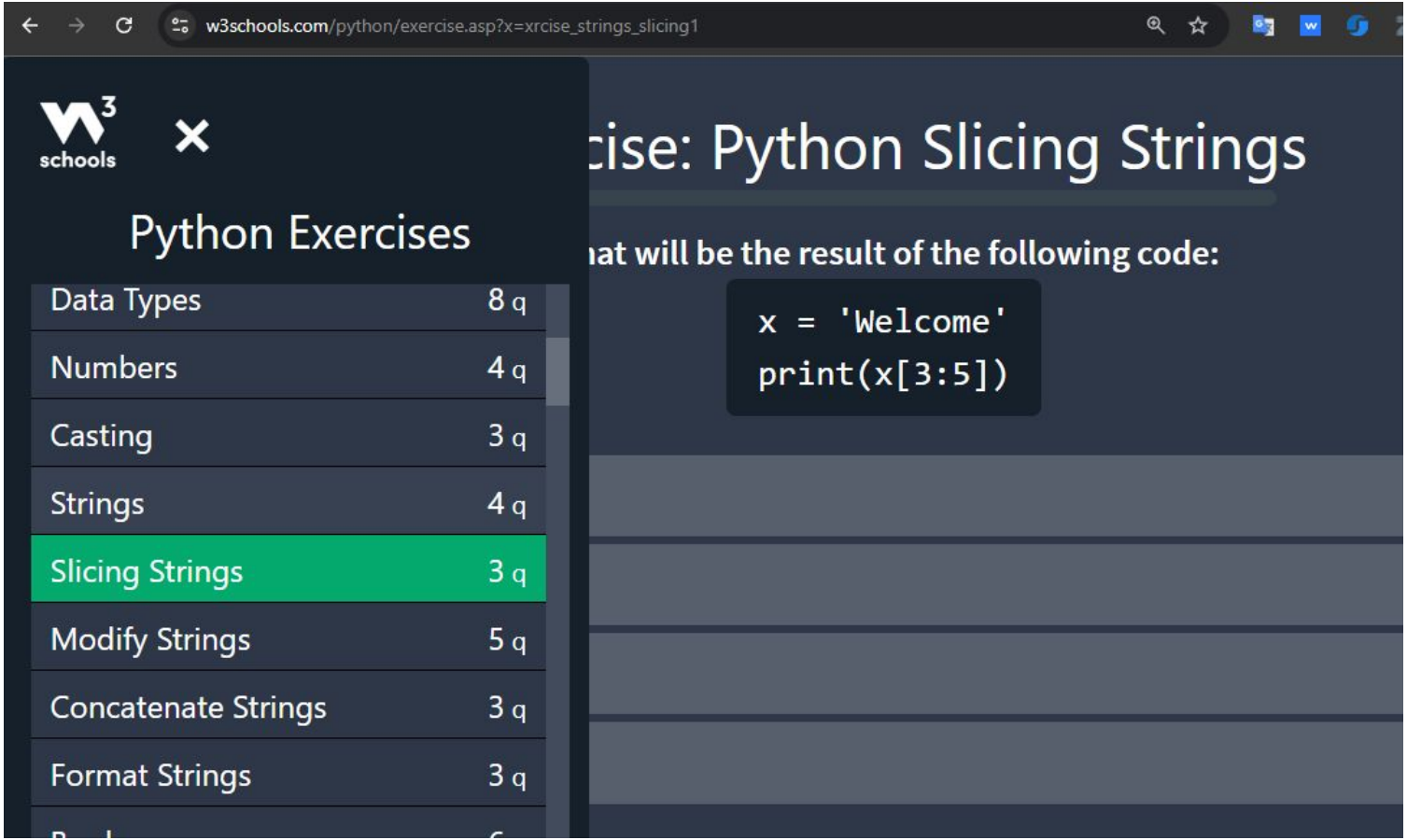
El objetivo de este ejercicio es simular la generación de un recibo o una factura. Debes usar **f-strings** para incrustar variables y realizar un formateo numérico sencillo.

Una pequeña cafetería necesita automatizar la impresión de sus recibos. Tienes todas las variables listas, solo necesitas combinarlas de forma legible.

Define las siguientes variables al inicio del ejercicio:

```
# Variables del Producto

producto = "Espresso Doble"

precio_unitario = 3.50

# Variables de la Transacción

cantidad = 3

tasa_iva = 0.16 # 16% de IVA

# Variables del Cliente

cliente_nombre = "Ana María López"

id_transaccion = "CAF-2025-472"
```

Autoevalúa tus conocimientos



Haz los 5 bloques de ejercicios de los Strings

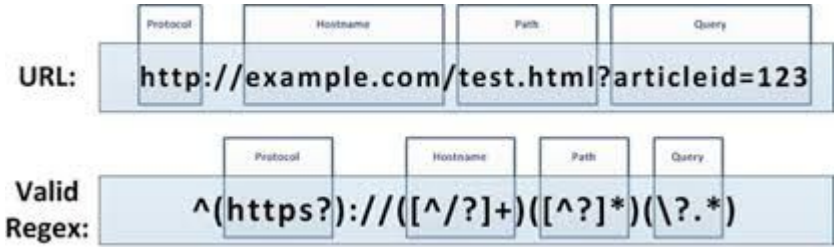
¡El poder de la BÚSQUEDA INTELIGENTE!

¿Cómo Encuentra Google lo que Buscas?

Expresiones Regulares (RegEx o Regexp)

RegEx: El Lenguaje para Hablar con el Texto

- Las **Expresiones Regulares** son secuencias de caracteres que forman un **patrón de búsqueda**.
- Son universales (Python, JavaScript, bases de datos...).
- En Python, usamos el módulo `re`.



Manos a la obra: El módulo re


```python
import re
# La función más usada: re.search()
resultado = re.search(patron, texto_fuente)
```

Conceptos clave del módulo:

- **`re.search(patrón, texto)`:** Busca el patrón **en cualquier parte** del texto.
- **`re.findall(patrón, texto)`:** Encuentra **todas** las coincidencias y las devuelve como una lista.
- **`re.match(patrón, texto)`:** Solo busca si el patrón está al **inicio** del texto.

Patrones Básicos (Los "Comodines")

Carácter	Nombre	Significado	Ejemplo de Uso
`.`	Punto	**Cualquier** carácter (excepto salto de línea).	`h.la` busca "hola", "hela", "hila"...
`\d`	Dígito	Cualquier **dígito** (0-9).	`\d\d` busca "12", "05", "99"...
`\w`	Palabra	Cualquier carácter de **palabra** (letras, números, _).	`\w\w\w` busca "sol", "a1b", "pro"...
`\s`	Espacio	Cualquier carácter de **espacio en blanco** (espacio, tabulador, etc.).	`hola\sadios` busca "hola adios"

Nota: Las versiones en mayúscula (`\D`, `\W`, `\S`) son la negación (lo opuesto).

Cuantificadores (¿Cuántas Veces?)

Carácter	Nombre	Significado	Ejemplo de Uso
*	Cero o Más	Cero o más veces el carácter anterior.	a*b busca "b", "ab", "aab", "aaab"...
+	Uno o Más	Uno o más veces el carácter anterior.	a+b busca "ab", "aab", "aaab"... (no busca "b")
?	Cero o Uno	Cero o una vez el carácter anterior (opcional).	colo(u)?r busca "color" y "colour"
{n}	Repetición	Exactamente n veces.	\d{4} busca un número de **4 dígitos**.
{n,m}	Rango	Entre n y m veces (mínimo n, máximo m).	\d{3,5} busca entre **3 y 5 dígitos**.

Ejemplos en Acción (Usando re.findall)

Ejemplo 1: Buscar números de 4 dígitos (Año)

```
texto = "Nació en 1985 y el libro es de 2023."
patron = r'\d{4}' # Busca exactamente 4 dígitos
print(re.findall(patron, texto))
# Salida: ['1985', '2023']
```

Ejemplo 2: Buscar una palabra que empieza por 'a' y acaba por 'o'

```
texto = "animal, árbol, amigo, oso."
patron = r'a\w*o' # 'a' + 0 o más letras/números + 'o'
print(re.findall(patron, texto))
# Salida: ['animalo', 'amigo'] -> ¡Ojo, incluye 'animalo'! (Discusión
sobre límites)
```

Nota: Para introducir una expresión regular en Python usamos comillas simples y delante una letra r

Ejercicio 27: El Desafío del Hacker Ético

¡Somos un equipo de ciberseguridad! Nuestra misión es encontrar "datos sensibles" en un gran texto usando solo expresiones regulares para validarlos. Texto a usar (cópialo):

```
data = """
Usuario: perez.juan
Email de contacto: juan.perez@dominio.com
Fecha de acceso: 15-09-2023
Código de cliente: ABC123456
Teléfono de emergencia: 601 234 567
Archivos encontrados: a_1.txt, b_22.py, c_333.pdf, d_4444.jpg
"""
```

Tarea	Patrón RegEx	Función a usar
A) ** Buscar todos los **emails (tienen el patrón `@` y `.`).		
B) ** Buscar las **fechas con formato **DD-MM-AAAA** (dos dígitos, guion, dos dígitos, guion, cuatro dígitos).		
C) ** Buscar el **Código de cliente (siempre empieza por **3 letras mayúsculas** y luego **6 dígitos**).		
D) ** Buscar todos los **nombres de archivo que terminan en **`.py`**.		