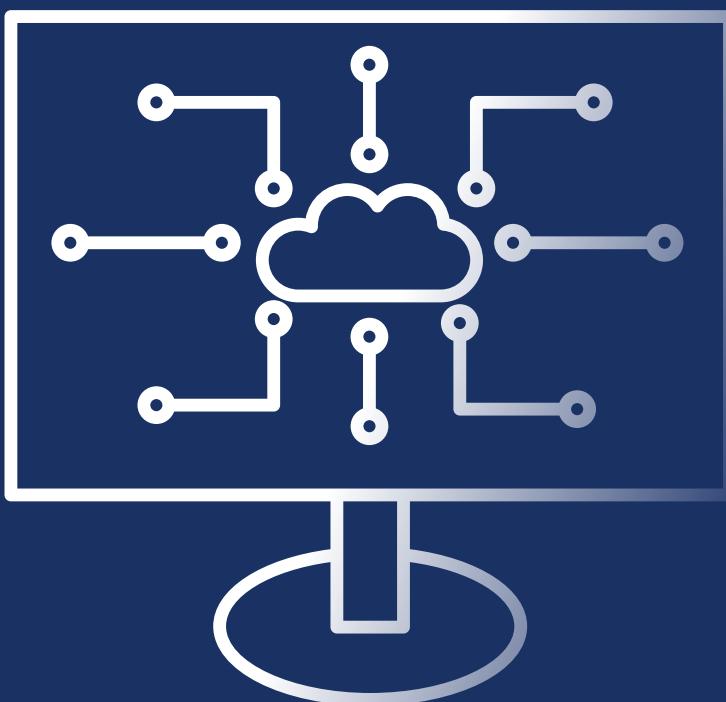


# Programación I



---

Lic. Julia Monasterio  
[marmonasterio@uade.com.ar](mailto:marmonasterio@uade.com.ar)



# Unidades

-  **Unidad 1:** Repaso de matrices, estructura, operaciones y uso en funciones
-  **Unidad 2:** Implementación y gestión en proyectos en Git
-  **Unidad 3:** Listas avanzadas, cadena de caracteres y expresiones regulares.
-  **Unidad 4:** Manipulación avanzada de diccionarios, tuplas y conjuntos.
-  **Unidad 5:** Excepciones y pruebas unitarias
-  **Unidad 6:** Archivos
-  **Unidad 7:** Recursividad

# Clase N°3

## TEMAS

- Métodos de lista
- Segmentación de listas

# Operaciones con listas

# OPERACIONES CON LISTAS

- La función **len()** devuelve la cantidad de elementos de una lista

**lista = [3,4,5,6]**

**print(len(lista)) --> #4**

# OPERACIONES CON LISTAS

- La función **sum()** devuelve la suma de los elementos de la lista

lista = [3,4,5,6]

```
print(sum(lista)) --> 18
```

- La lista no debe contener números

# Práctica en clase

Crear un grafico de barras con los porcentajes obtenidos por cada candidato en una elección



# Función para cargar la lista de votos por candidato

```
def cargarListaVotos():
    """Funcion que va a permitir cargar los votos de cada
    candidato. Hasta que el usuario presione -1"""

    votos=[]
    bandera=True

    while bandera:
        votosCandidato = int(input("Ingrese la cantidad de votos del candidato.
        if votosCandidato== -1:
            bandera=False
        else:
            votos.append(votosCandidato)
    return votos
```

# Función para calcular porcentajes de los votos

```
def calcularPorcentajeCandidato(listaVotos):
    """Funcion que recibe una lista con la cantidad nominal de votos
    de cada candidato, y calcula el porcentaje.
    Devuelve una lista con los porcentajes de cada candidato"""
    total=sum(listaVotos)
    porcentajes=[]
    if total!=0:
        for i in range(len(listaVotos)):
            porcentajeCandidato = listaVotos[i]*100/total
            porcentajes.append(porcentajeCandidato)
    return porcentajes
```

# Funcion para imprimir los resultados

```
def imprimirResultados(listaVotos, porcentajes):
    for i in range(len(listaVotos)):
        print("- Candidato %d: %d votos (%.2f%%)"%(i+1,listaVotos[i], porcentajes[i]), end=" ")
        for j in range(int(porcentajes[i]/10)):
            print("*", end="")
        print()
```

# ESPECIFICADORES DE CONVERSIÓN

- Son símbolos que se utilizan en las cadenas de formato para indicar como se deben convertir y formatear los valores antes de insertarlos en una cadena de texto.
- Se utiliza el símbolo %
- Los motivos principales donde se pueden usar son:
  - Impresión en consola
  - Formateo de cadenas
  - Escritura de archivos
  - Construcción de consultas SQL (aunque no es recomendado)

# ESPECIFICADORES DE CONVERSION

- **%s** inserta cualquier tipo de valor en la cadena de formato como una cadena de texto
- Se utiliza el % entre la cadena y el valor que se desea insertar

```
nombre="Pepe"
```

```
print("Tu nombre es %s" % nombre)
```

# ESPECIFICADORES DE CONVERSION

- **%d** inserta un numero entero en la cadena

```
edad=34  
print("Tu edad es %d" % edad)
```

- **%f** inserta un numero decimal en la cadena

```
PI=3.14159  
  
print("El valor de PI es %f"% PI)
```

# ESPECIFICADORES DE CONVERSION

- `%%` insertar un signo porcentaje. Se usa cuando se necesita insertar el literal de %

```
IVA=21
```

```
print("El porcentaje de IVA es %d %%"% IVA)
```

```
El porcentaje de IVA es 21 %
```

# F-STRING

- Las **formatted-string** indica que la cadena puede incluir expresiones formateadas que se evaluaran en tiempo de ejecucion.
- Se debe colocar primero la letra **f** y luego encerrar entre llaves {} la variable a mostrar.
- Si se desea dar un formato determinado a la variable se debe colocar luego de la misma **:formato**

```
IVA=21
```

```
print(f"El porcentaje de IVA es {IVA:.2f} %")
```

```
El porcentaje de IVA es 21.00 %
```

# F-STRING

- Otros especificadores de formato pueden ser
  - :d para enteros
  - :s para cadena de caracteres

# OPERACIONES CON LISTAS

- La función **max()** devuelve el mayor de los elementos de la lista

lista = [4,6,3,5]

print(max(lista)) --> 6

- La lista no debe contener elementos homogéneos

# OPERACIONES CON LISTAS

- La función **min()** devuelve el menor de los elementos de la lista

lista = [4,6,3,5]

print(min(lista)) --> 4

- **La lista no debe contener elementos homogéneos**

# OPERACIONES CON LISTAS

- Las funciones **min()** y **max()** también pueden ser utilizadas con un conjunto de valores, constantes o variables

```
print(max(5,4,8,3,45,43))
```

```
print(max('PEPE','LOPEZ','ZOE','ZARA'))
```

```
print(min(5,4,8,3,45,43))
```

```
print(min('PEPE','LOPEZ','ZOE','ZARA'))
```

```
45  
ZOE  
3  
LOPEZ
```

# OPERACIONES CON LISTAS

- La función **list()** convierte cualquier iterable en una lista. También lo podemos utilizar para crear una lista vacía.

lista = **list(range(5))** -> va a crear una lista [0,1,2,3,4]

- Se puede usar con rangos, cadenas, tuplas, conjuntos.

# OPERACIONES CON LISTAS

- Ejemplos de List

```
lista=list()  
  
nombre="Juan"  
  
listita=list(nombre)  
  
rango= list(range(5))
```

```
[]  
[ 'J', 'u', 'a', 'n' ]  
[0, 1, 2, 3, 4]
```

# OPERACIONES CON LISTAS

- El operador **in** permite verificar la presencia de un elemento
- **Devuelve True o False**

```
lista=[1,2,3,4]

if 4 in lista:
    print("Existe el 4 en la lista")
```

Existe el 4 en la lista

# OPERACIONES CON LISTAS

- Otro ejemplo:

```
frutas=["manzana","pera","banana"]

busqueda=["pera","manzana"]

for fruta in busqueda:
    if fruta in frutas:
        print(f"{fruta} se encuentra en la lista de frutas")
    else:
        print(f"{fruta} no se encuentra en la lista de frutas")
```

pera se encuentra en la lista de frutas  
manzana se encuentra en la lista de frutas

# OPERACIONES CON LISTAS

- El operador **not in** comprueba la ausencia de un elemento
- **Devuelve True o False**

```
lista=[1,2,3,4]

if 7 not in lista:
    print("NO existe el 7 en la lista")
```

NO existe el 7 en la lista

# Práctica en clase

Escribir una función que reciba como parámetros dos números correspondientes al mes y año de una fecha y devuelva cuantos días tiene ese mes en ese año



# Función para obtener cantidad de días

```
def obtenerCantDias(mes,anio):
    """Funcion para obtener la cantidad de dias a partir de
    mes y anio"""
    if mes in [1,3,5,7,8,10,12]:
        dias=31
    elif mes in [4,6,9,11]:
        dias=30
    elif mes==2:
        #Anio bisiesto si es modulo de 4 y no de 100
        #Excepto que sea modulo de 400
        if(anio%4==0 and anio%100!=0) or (anio%400==0):
            dias=29
        else:
            dias=28
    else:
        dias=-1
    return dias

#Programa Principal
print(obtenerCantDias(14,2024))
```

# Métodos con listas

# Métodos

- Un **método** es una función que pertenece a un objeto
- Para **Python** todos los tipos de datos son **objetos**
- **Los métodos permiten manipular los datos almacenados en el objeto**
- **Se escriben luego del nombre del objeto, separados por un punto**

# METODOS CON LISTAS

- El metodo **append(<elem>)** agrega un elemento al final de una lista

```
lista=[4,5,6,7]
```

```
lista.append(8)
```

```
print(lista)
```

```
[4, 5, 6, 7, 8]
```

# METODOS CON LISTAS

- El método **insert(<pos>,<elem>)** inserta un elemento en la lista, en una posición determinada

```
definicion... • Cursos Beiguanos • Clas...  
lista=[4,5,6,7]  
  
lista.insert(2,17)  
lista.insert(0,13)  
  
print(lista)
```

```
[13, 4, 5, 17, 6, 7]
```

# METODOS CON LISTAS

- El método **pop(<pos>)** elimina y devuelve un elemento de la lista, identificando su posición.
- Si no se envía posición se elimina el ultimo elemento de la lista
- Da error si la posición esta fuera de rango
- Devuelve el elemento que elimina

# METODOS CON LISTAS

- Ejemplos:

```
lista=[4,5,6,7]

lista.pop(0)

print(lista)

lista.pop()

print(lista)
```

```
[5, 6, 7]
[5, 6]
```

```
lista=[4,5,6,7]

elemento=lista.pop(3)

print(elemento)
```

```
7
```

# METODOS CON LISTAS

El método **remove(<elem>)** elimina la primera aparición de un elemento en la lista, identificado por su valor.

```
lista=[4,5,6,7,4]  
  
elemento=lista.remove(4)  
  
print(lista)
```

```
[5, 6, 7, 4]
```

- Devuelve un error si el elemento no existe

```
lista=[4,5,6,7,4]  
elemento=lista.remove(9)
```

```
elemento=lista.remove(9)  
^^^^^^^^^^^^  
ValueError: list.remove(x): x not in list
```

# METODOS CON LISTAS

- pero... si quiero eliminar todas las ocurrencias?



```
lista=[4,5,6,7,4]
for elemento in lista:
    lista.remove(4)
print(lista)
```

- Este código da error porque se esta modificando la lista mientras la estamos recorriendo

# METODOS CON LISTAS

- pero... si quiero eliminar todas las ocurrencias?



```
while 4 in lista:  
    lista.remove(4)  
  
print(lista)
```

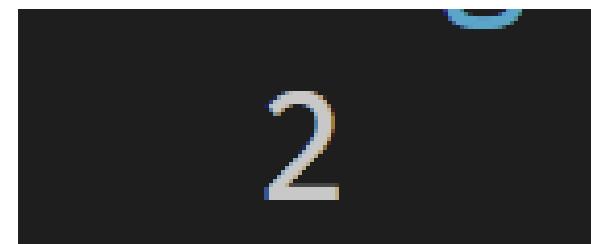
# METODOS CON LISTAS

El método **index(<elem>)** busca un elemento y devuelve su posición

- Provoca un error si no lo encuentra

```
lista=[5,6,7,8,9]

indice = lista.index(7)
|
print(indice)
```



2

# METODOS CON LISTAS

Tambien se puede utilizar con dos o tres parametros:

- lista.index(5,2) -> busca el valor 5 a partir del segundo subindice
- lista.index(5,2,6) -> busca el valor 5 desde el segundo subindice hasta el anteultimo subindice. **Es decir el valor final no esta incluido en el intervalo de busqueda**

```
lista=[5,6,7,8,9]  
  
indice = lista.index(7,0,3)
```

```
lista=[5,6,7,8,9]  
  
indice = lista.index(7,2)
```

# METODOS CON LISTAS

El método **count(<elem>)** devuelve la cantidad de repeticiones de un elemento.

- Devuelve 0 si no lo encuentra

```
lista=[5,6,7,8,9,8]  
  
indice = lista.count(8)  
  
print(indice)
```

2

```
lista=[5,6,7,8,9,8]  
  
indice = lista.count(10)  
  
print(indice)
```

0

# METODOS CON LISTAS

El método **clear()** elimina todos los elementos de una lista, **pero no elimina la lista**

```
lista=[5,6,7,8,9,8]  
  
indice = lista.clear()  
  
print(lista)
```

```
[ ]
```

# METODOS CON LISTAS

El método **reverse()** invierte los elementos de la lista. Trabaja sobre la misma lista, es decir no crea una lista nueva

```
lista=[5,6,7,8,9,8]
```

```
lista.reverse()
```

```
print(lista)
```

```
[8, 9, 8, 7, 6, 5]
```

# METODOS CON LISTAS

El método **reversed()** invierte los elementos de la lista y crea una nueva lista con los elementos invertidos

- Si es necesario invertir una lista sin modificar la lista original, es posible utilizar la función **reversed()**. Se debe utilizar en conjunto con el **list()** para obtener una nueva lista invertida. Si no utilizamos list nos devolverá un objeto de tipo **reversed()**

```
lista=[7,8,9,10,11,12]

lista2=list(reversed(lista))

print(lista)
[7, 8, 9, 10, 11, 12]

print(lista2)
```

```
[7, 8, 9, 10, 11, 12]
[12, 11, 10, 9, 8, 7]
```

# METODOS CON LISTAS

El método **sort()** ordena los elementos de la lista. Trabaja sobre la misma lista, es decir no crea una lista nueva

- Ordena de menor a mayor, utilizando cada valor como criterio de comparacion

```
lista=[5,6,7,8,9,8]  
  
lista.sort()  
  
print(lista)
```

```
[5, 6, 7, 8, 8, 9]
```

# METODOS CON LISTAS

...pero si quiero ordenarla de mayor a menor.

Para eso al metodo **sort** le pasamos como parametro **reverse=True**

```
lista=[5,6,7,8,9,8]  
lista.sort(reverse=True)  
print(lista)
```

```
[9, 8, 8, 7, 6, 5]
```

Se pueden hacer mucho mas cosas con **sort** que ya lo veremos cuando veamos funciones lambda

# METODOS CON LISTAS

La función **sorted()** permite ordenar cualquier iterable (rango, lista, string, tupla, conjunto, diccionario)

- Devuelve una copia ordenada

<b>sort()</b>	<b>sorted()</b>
Modifica la lista original	Devuelve una nueva lista
Solo soporta lista	Cualquier iterable

# METODOS CON LISTAS

El método **copy()** permite crear una nueva lista a partir de una lista existente.

- Para invocarla se debe utilizar **<nombre\_lista\_original>.copy()**

```
luis@luis-MX-OptiPlex-5070:~/Desktop$ python3
[...]
lista= ["PEPE", "JUAN", "CARLOS"]

lista2=lista.copy()

lista.append("SUSANA")

print(lista)
print(lista2)
```

```
[...]
['PEPE', 'JUAN', 'CARLOS', 'SUSANA']
['PEPE', 'JUAN', 'CARLOS']
```

# METODOS CON LISTAS

pero... porque no utilizamos el = ??

Que sucede en el siguiente caso?

```
lista= ["PEPE", "JUAN", "CARLOS"]  
  
lista2=lista  
  
lista.append("SUSANA")  
  
print(lista)  
print(lista2)
```



# METODOS CON LISTAS

## APPEND vs EXTEND

```
lista=['a','b','c','d']
```

```
lista2=[6,7]
```

```
lista.append(lista2)
```

```
print(lista)
```

```
['a', 'b', 'c', 'd', [6, 7]]
```

```
lista=['a','b','c','d']
```

```
lista2=[6,7]
```

```
lista.extend(lista2)
```

```
print(lista)
```

```
['a', 'b', 'c', 'd', 6, 7]
```

# Numeros al azar

# Números al azar

- Son numeros generados por la computadora
- Se utilizan cuando se requiere un factor de azar
- **Python** tiene varias funciones relacionadas con ello, **todas pertenecen al modulo random.**

Por eso se debe realizar **import random**

# NUMEROS AL AZAR

- **random.random()** devuelve un numero real dentro del intervalo de 0 a 1. Sin incluir el 1.

```
import random  
  
numero=random.random()  
  
print(numero)
```

0.7924192427779038

# NUMEROS AL AZAR

- **random.randint(<min>,<max>):** devuelve un numero entero al azar dentro del intervalo dado. Si incluye el max.

```
import random

numero=random.randint(4,8)

print(numero)
```

8

# NUMEROS AL AZAR

- **random.choice(<secuencia>):** devuelve un elemento elegido al azar dentro de una secuencia pasada como parámetro.
- La secuencia puede ser una lista, un string, una tupla, un rango.

```
import random
opciones=["Piedra","Papel","Tijera"]

situacion = random.choice(opciones)

print(situacion)
```

Piedra

# NUMEROS AL AZAR

- **random.choice(<secuencia>):** devuelve un elemento elegido al azar dentro de una secuencia pasada como parámetro.
- La secuencia puede ser una lista, un string, una tupla, un rango.

```
import random
opciones=["Piedra","Papel","Tijera"]

situacion = random.choice(opciones)

print(situacion)
```

Piedra

# NUMEROS AL AZAR

- **random.shuffle(<lista>):** mezcla los elementos de una lista (no retorna una nueva). Es decir que altera la posición de los mismos.

```
import random

lista=["Espada","Basto","Copas","Oros"]

random.shuffle(lista)

print(lista)
```

['Basto', 'Copas', 'Oros', 'Espada']

# Práctica en clase

Para un juego de generala se necesita desarrollar una función que simule el lanzamiento de los cinco dados.



# Función generala

```
import random

def lanzarDados(cuantos):
    dados= []

    for i in range(cuantos):
        dados.append(random.randint(1,6))
    return dados
```

#Programa Principal

jugada=lanzarDados(5)

print(jugada)

# Segmentacion de Listas

# Rebanadas o List Slicing

- Una **rebanada o slice** es una manera de referirse a un grupo de elementos pertenecientes a una lista
- En lugar de utilizar un solo subíndice se utilizan dos o tres, separados por : **dos puntos**
- La segmentación de listas devuelve una lista nueva a partir de una existente sin alterar la original.

# Rebanadas o List Slicing

- Ejemplo:

```
lista=[7,8,9,10,11,12]
```

```
sublista=lista[2:5]
```

```
print(sublista)
```

```
[9, 10, 11]
```

- Los subíndices indican el inicio y fin del slice o rebanada
- El subíndice final **no está incluido**

# Rebanadas o List Slicing

- Otros ejemplos:

```
lista=[7,8,9,10,11,12]
```

```
sublista=lista[3:]
```

```
print(sublista)
```

```
[10, 11, 12]
```

```
lista=[7,8,9,10,11,12]
```

```
sublista=lista[:3]
```

```
print(sublista)
```

```
[7, 8, 9]
```

- Dejar en blanco alguno de los subíndices hace que se considere el extremo correspondiente de la lista (ya sea inicio o final)

# Rebanadas o List Slicing

- Otros ejemplos:

```
lista=[7,8,9,10,11,12]  
  
sublista=lista[1:6:2]  
  
print(sublista)
```

```
[8, 10, 12]
```

- Cuando se usan tres subíndices, el tercero actúa como **incremento**. En este ejemplo arma una lista del 1 al 6 (sin incluir el 6), saltando de 2 en 2.

# Rebanadas o List Slicing

- Otros ejemplos:

```
lista=[7,8,9,10,11,12]
```

```
sublista=lista[::-1]
```

```
print(sublista)
```

```
[12, 11, 10, 9, 8, 7]
```

- Un incremento negativo toma los elementos de atrás hacia delante

# Rebanadas o List Slicing

- Las rebanadas también funcionan con variables.

```
lista=[7,8,9,10,11,12]
a=3
b=5

sublista=lista[a:b]

print(sublista)
```

```
[10, 11]
```

# Rebanadas o List Slicing

- Una rebanada **nula** es una rebanada o slice que no contiene ningún elemento
- Se crea utilizando el mismo subíndice para inicio y fin.
- No se utiliza casi nunca, pero nos sirve para insertar un nuevo elemento en alguna posición específica

```
lista=['a','b','c','d']
```

```
lista[2:2]='Y'
```

```
lista[2:2]=[6,7]
```

```
print(lista)
```

```
['a', 'b', 6, 7, 'Y', 'c', 'd']
```

# Práctica en clase

Obtener sublistas con los primeros y últimos N elementos de una lista



# Utilizacion de rebanadas

```
lista= list(range(10))

n=int(input("Cuantos elementos desea tomar.?"))

comienzo=lista[:n]
final= lista[n:]

print("Lista Original: ",lista)
print("Comienzo de lista desde cero hasta elemento elegido: ",comienzo)
print("Final de lista, desde el elemento elegido hasta el final: ",final)
```

```
Cuantos elementos desea tomar.?4
Lista Original: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Comienzo de lista desde cero hasta elemento elegido: [0, 1, 2, 3]
Final de lista, desde el elemento elegido hasta el final: [4, 5, 6, 7, 8, 9]
```

# Resumen de la clase

- Operaciones len, min, max, sum
- Especificadores de conversión - Format String
- Funcion list
- Operador in, not in
- Metodos append, insert, pop, remove, index, count, clear, reverse, reversed, sort, sorted, copy, extend
- Numeros al azar
- Segmentacion de listas

Muchas gracias!

Consultas?

