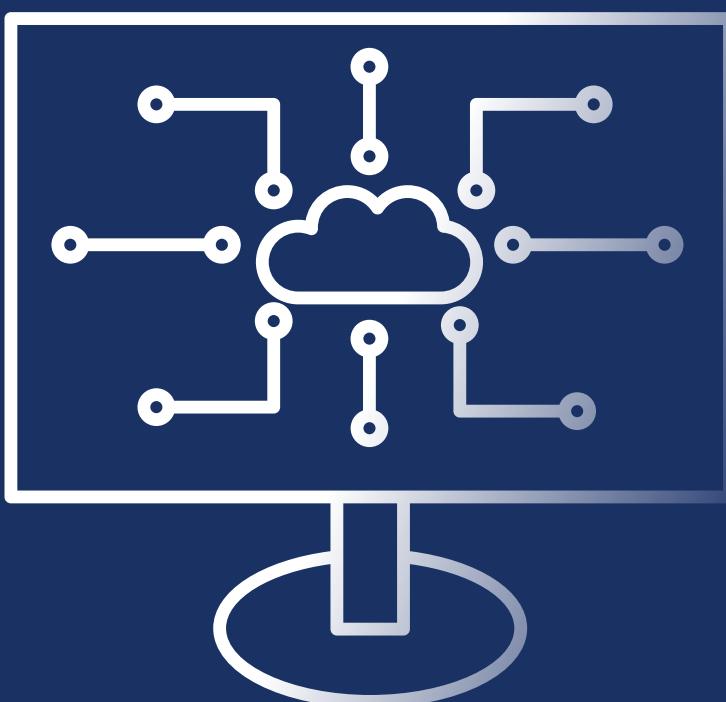


Programación I



Lic. Julia Monasterio
marmonasterio@uade.com.ar

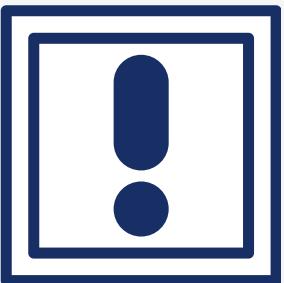




Lugar: Aula 348 - Montserrat



Día/Horario: Martes- 18:30 a 22



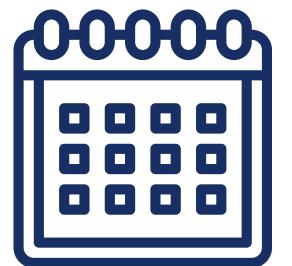
75% de Asistencia



NO PROMOCIONABLE

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



Fechas Importantes

31/8/2024 - Clase Virtual - 9 a 13

24/9/2024 - Primer preentrega de TP (40%)

19/11/2024 - Segunda preentrega de TP (80%)

3/12/2024 - Entrega final de TP



Modalidad de Evaluación

Proyecto Cuatrimestral por etapas

El proyecto cuatrimestral tiene como objetivo principal que los estudiantes apliquen los conocimientos adquiridos en Programación I mediante el desarrollo de un proyecto cercano a una implementación laboral real.

Esto promueve la comprensión profunda de los conceptos, el trabajo colaborativo y el desarrollo de habilidades prácticas en programación.



Desarrollo del proyecto

- Los estudiantes deberán proponer un tema (puede ser un juego, programa de calculo de valores, o cualquier otro tema) sujeto a la aprobación del docente.
- Los proyectos se programarán con el lenguaje Python y deben cubrir todos los temas del programa de la asignatura.
- Los estudiantes se organizarán en equipos de 3 o 4 miembros, siendo responsables de la planificación, ejecución y presentación del proyecto.

No se puede realizar el TP de forma individual



Entregas del proyecto

- **1era Entrega:** Se espera que en esta entrega se un avance del 40% del trabajo final.
- **2da Entrega:** Se espera que en esta entrega se un avance del 80% del trabajo
- **3ra Entrega Final:** Se espera que en esta entrega sea el 100% del Proyecto



Evaluación del proyecto

- **Se evaluarán los siguientes aspectos:**
 - Funcionamiento del programa.
 - Cumplimiento de los objetivos planteados.
 - Técnicas de programación y estrategias de resolución del problema.
 - Prolijidad y claridad del código y de la interfaz de usuario.
 - Se valorarán las presentaciones que se realicen en formato PowerPoint u otro a elección del grupo.
 - **La calificación final recibida por cada integrante dependerá no solo del trabajo presentado sino también de la exposición ejercida en forma individual, así como las preguntas que se le realizaran a cada estudiante posteriormente a la exposición del trabajo. Por esta razón, pueden tener notas diferentes miembros del mismo grupo.**



Evaluación constante - Todas las clases



Defensa de los entregables

- **Metodología para la defensa:**

- Se deberá ejecutar el código sin errores, recorriendo todas las opciones o casuísticas que requieran.
- Deberán contar cuales fueron las técnicas utilizadas y las estrategias de resolución, como así también los desafíos que encontraron durante el proceso.
No es objetivo LEER todas las líneas de código, sino a nivel conceptual que es lo que cada parte del programa está realizando o representando de la solución.
- Todos los miembros del equipo deberán exponer parte del trabajo.
- Deben tener un soporte visual, presentación PPT o similar, a criterio del equipo.



ACLARACIONES

- No se aceptarán trabajos que incluyan funciones o herramientas de Python no tratados en clase. (salvo que exista un previo pedido y autorización por parte del docente)
- El trabajo para exponer deberá ser el mismo que se designó el día de la fecha de inicio (segunda clase).

INTRODUCCIÓN

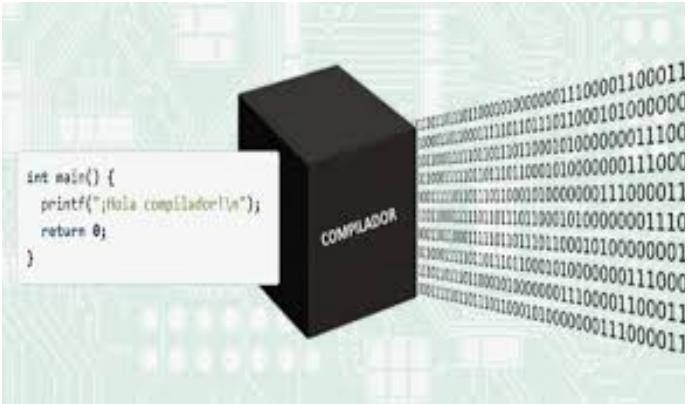


- En la materia continuaremos trabajando en el lenguaje de programación Python
- Como editor podran utilizar el que prefieran (Thony, Visual Studio Code)



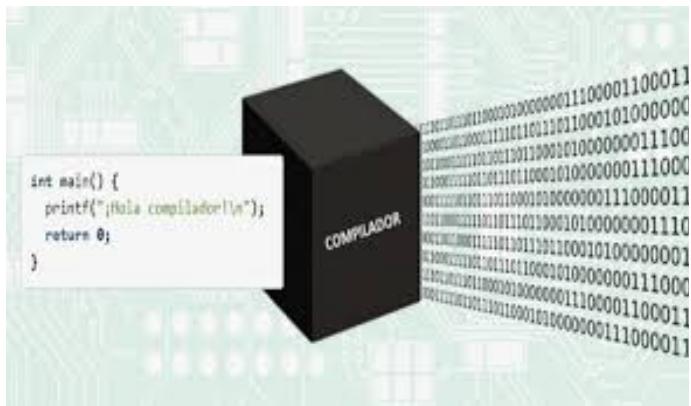
PYTHON

- Es un lenguaje de alto nivel
- Es interpretado. No se compila
- Es multiparadigma
- Es portable
- Es sensible a mayúsculas y minúsculas

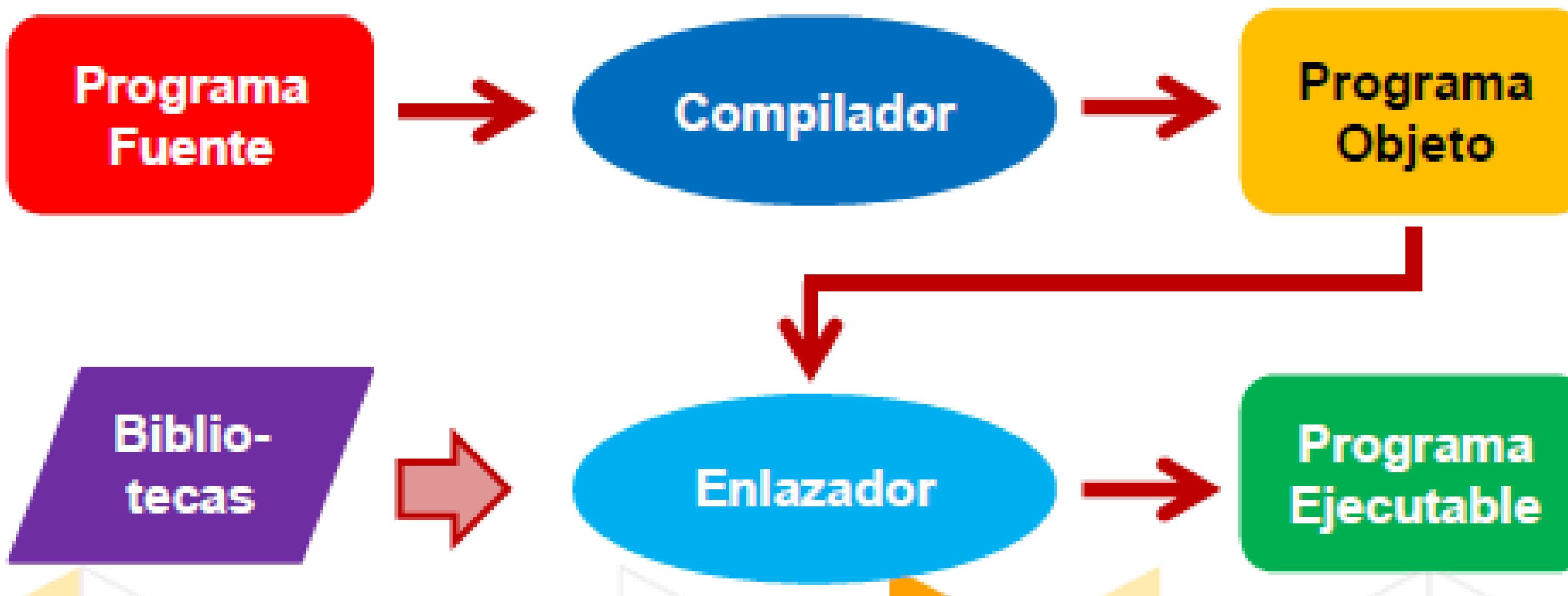


COMPILADOR

- Convierte el programa fuente, escrito en un **lenguaje de alto nivel**, en un **programa objeto** es decir en un lenguaje maquina
 - Este programa objeto debe ser enlazado o vinculado con bibliotecas proporcionadas por el fabricante creando así el **programa ejecutable**



COMPIILADOR

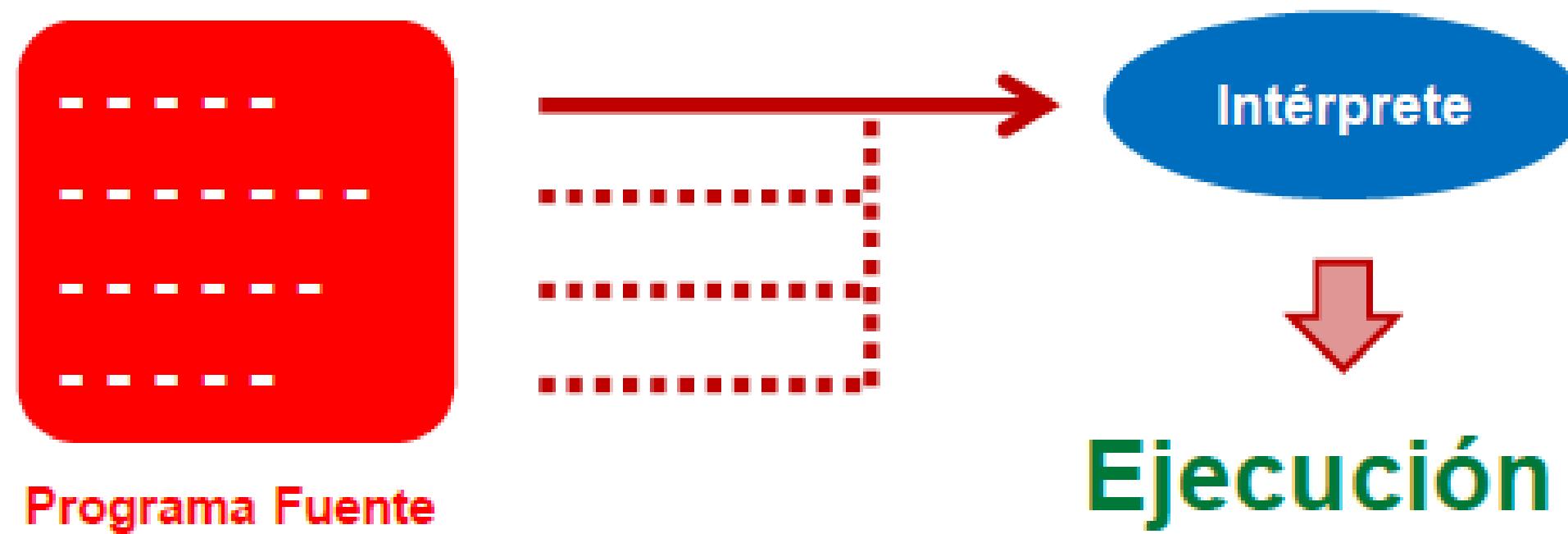




INTERPRETE

- Traduce una por una las líneas del programa fuente, ejecutándolas inmediatamente

Proceso de Interpretación



COMPILADOR vs INTERPRETE

- Los programas interpretados suelen ser más lentos que los programas compilados
- Tienen menos requisitos sintácticos que los compiladores, lo que permite probar los programas a medida que se van desarrollando

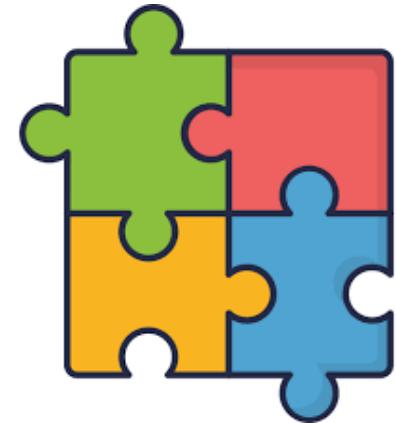
Clase N°1

TEMAS

- Repaso de funciones y listas
- Matrices como arreglos bidimensionales de elementos.
- Uso de índices para acceder a componentes específicos y recorrer la matriz.
- Operaciones comunes incluyen la suma, resta, multiplicación por un escalar, multiplicación de matrices, y la transposición.
- Implementación del pasaje de matrices como parámetros en funciones para manipulaciones específicas.

REPASO FUNCIONES



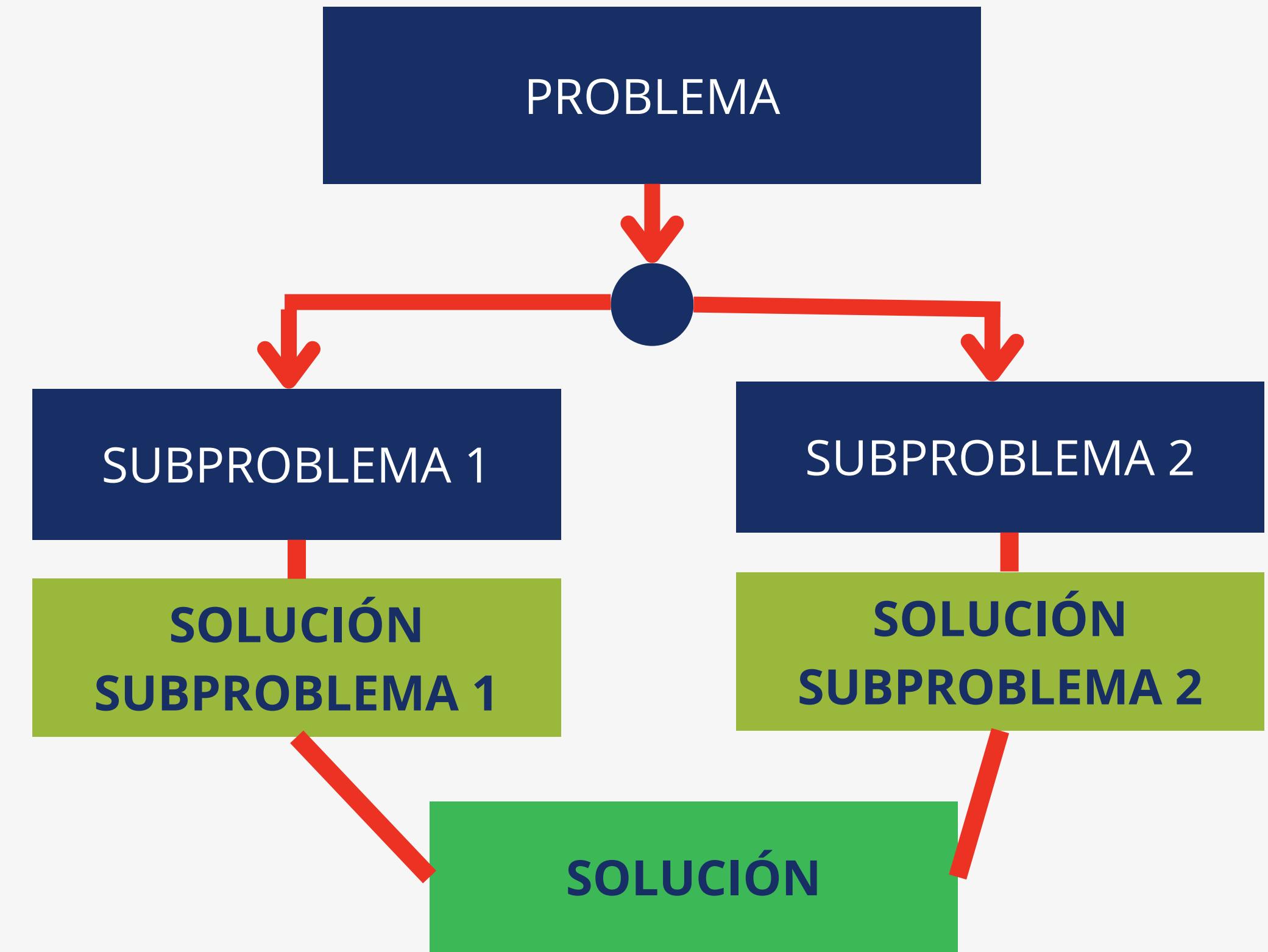


MODULARIZACIÓN

- Se divide el programa en muchos componentes o bloques pequeños autónomos llamados módulos, que son manejables, lógicos y funcionales.
- El módulo es un bloque de sentencias que resuelve un problema particular. Cada módulo contiene todo lo necesario para cumplir con su propia funcionalidad y se puede editar o modificar sin que se vea afectado el resto del proyecto.
- Dependiendo del lenguaje de programación estos se denominan procedimientos, o también como subprogramas, rutinas o funciones.

MODULARIZACIÓN

DIVIDIR UN
PROBLEMA EN
PARTES MÁS
PEQUEÑAS





MODULARIZACIÓN

Código sin modularizar

```
num1 = 10
num2 = 20
suma1 = num1 + num2
print(f'La suma de {num1} y {num2} es {suma1}')
|
num3 = 30
num4 = 40
suma2 = num3 + num4
print(f'La suma de {num3} y {num4} es {suma2}')
```

Código modularizado

```
def sumar_y_mostrar(a, b):
    suma = a + b
    print(f'La suma de {a} y {b} es {suma}')

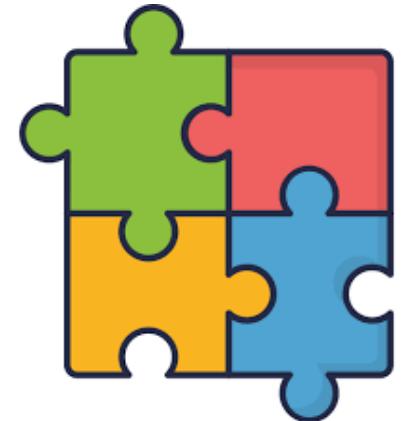
num1 = 10
num2 = 20
sumar_y_mostrar(num1, num2)

num3 = 30
num4 = 40
sumar_y_mostrar(num3, num4)
```



CARACTERISTICAS DE LOS MODULOS

- Deben tener un nombre único y representativo de la funcionalidad que representan
- No deben utilizar palabras claves en sus nombres, ni contener caracteres especiales
- **Los modulos deben tener una función específica y no excederse de sus responsabilidades designadas**

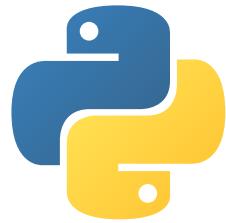


ENTRADA/SALIDA

CON
PARAMETROS
DE ENTRADA →
SIN PARAMETROS
DE ENTRADA →



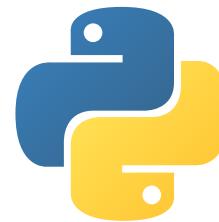
CON
DATOS DE SALIDA →
SIN DATOS
DE SALIDA →



FUNCIONES EN PYTHON

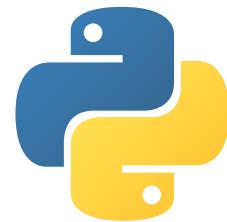
- Una función es un grupo de instrucciones que constituyen una unidad lógica del programa y resuelven un problema muy concreto.

```
def nombre_funcion (parametro1, parametro2, parametroN):  
    conjunto de instrucciones  
    return valor
```



FUNCIONES EN PYTHON

- **def** palabra reservada para definir una función. Toda función comienza con def
- **nombre_funcion** identificador
- **(parametro1, parametro2, parametroN)** lista de parámetros (opcional), es decir puede no recibir valores
- **return** palabra reservada para devolver un valor o resultado (opcional)
- El encabezado debe terminar con dos puntos (:)
- La sangría es obligatoria y determina el alcance de la función



NOMENCLATURA DE FUNCIONES

- Solo se permiten letras, números y guión bajo
- No pueden comenzar con un número
- No pueden coincidir con las palabras reservadas del lenguaje
- Es recomendable que el nombre de la función incluya un verbo infinitivo que describa su tarea
- Deben escribirse al principio del programa y luego el programa principal
- Es aconsejable que no sea el mismo nombre que una variable



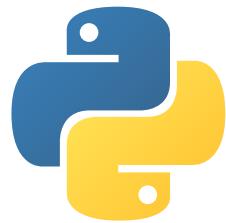
FUNCIONES EN PYTHON

```
def calcular_perimetro_cuadrado (lado):  
    perimetro= lado*4  
    return perimetro
```

¿CÓMO USAR O INVOCAR UNA FUNCIÓN?

Para invocar o utilizar una función se debe colocar el nombre de la función, y si tiene parámetros se debe enviar entre parentesis

```
calcular_perimetro_cuadrado(4)
```

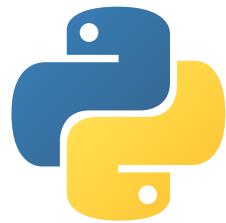


FUNCIONES EN PYTHON

DIFERENCIA ENTRE PARÁMETROS Y ARGUMENTOS

La función tiene definido un parámetro denominado **lado**.

Pero desde el código se invoca a la función **calcular_perímetro_cuadrado(4)**, se dice que se llama a la función con el argumento 4.

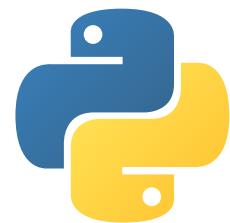


FUNCIONES EN PYTHON

ASIGNAR RESULTADO

```
resultado=calcular_perimetro_cuadrado(4)
```

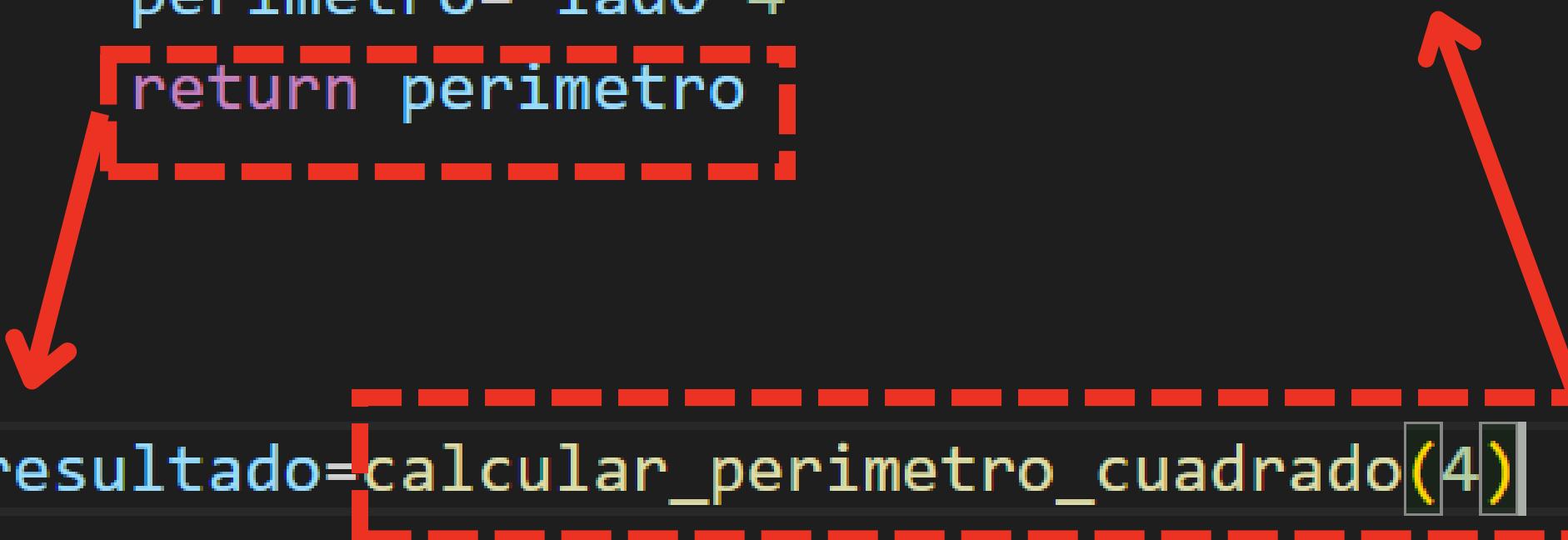
Como la función utiliza return, su resultado debe asignarse a una variable, o bien, ser utilizada como entrada en otra función.



INVOCACIÓN A FUNCIÓN

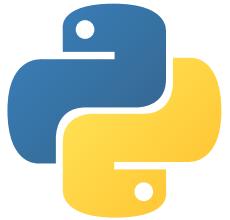
FUNCIONES EN PYTHON

```
def calcular_perimetro_cuadrado (lado):  
    perimetro= lado*4  
    return perimetro  
  
resultado=calcular_perimetro_cuadrado(4)
```



Cuando se encuentra el llamado de la función, el flujo de ejecución pasa a la primera instrucción de la función.

Cuando se llega a la última instrucción de la función, el flujo del programa sigue por la instrucción que hay a continuación de la llamada de la función.



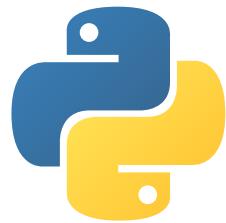
SENTENCIA RETURN

FUNCIONES EN PYTHON

La sentencia **return** hace que **termine** la ejecución de la función cuando aparece y el programa continúa por su flujo normal.

La sentencia **return** es **opcional**, puede devolver, o no, un valor y es posible que aparezca más de una vez dentro de una misma función.

Puede devolver más de un valor



FUNCIONES EN PYTHON

RETURN QUE NO RETORNA VALOR

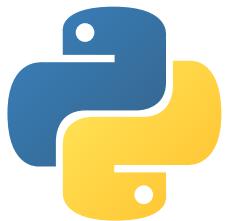
```
def esPar(numero):  
    if numero%2==0:  
        return True  
    else:  
        return
```

MÁS DE UN RETURN

```
def esPar(numero):  
    if numero%2==0:  
        return True  
    else:  
        return False
```

MÁS DE UN VALOR

```
def calcularCuadradoCubo(numero):  
    return numero**2, numero**3
```

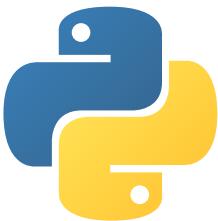


**EN PYTHON
UNA FUNCIÓN
SIEMPRE
DEVUELVE UN
VALOR**

FUNCIONES EN PYTHON

Incluso cuando no se especifica una sentencia return dentro de la función o la sentencia return no devuelve explícitamente ningún valor, la función devolverá implícitamente un valor predeterminado, que es **None**.

Esto puede ser explícitamente especificado utilizando la sentencia return con o sin un valor, o simplemente no especificándolo, en cuyo caso Python devolverá automáticamente **None**

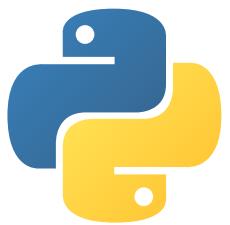


TIPO MUTABLES VS TIPOS INMUTABLES

FUNCIONES EN PYTHON

- **Tipos inmutables (e.g., int, float, str, tupla):** Se comportan como si fueran pasados por valor. Los objetos inmutables no pueden ser cambiados, por lo que parece que se pasó una copia.
- **Tipos mutables (e.g., lista, diccionario):** Se comportan como si fueran pasados por referencia.

Las funciones pueden modificar estos objetos, y los cambios se reflejarán fuera de la función.



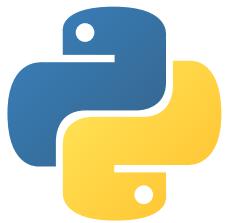
EJEMPLO INMUTABLE

FUNCIONES EN PYTHON

```
def modificar_valor(x):
    x = 10
    print(f'Valor dentro de la función: {x}')

a = 5
modificar_valor(a)
print(f'Valor fuera de la función: {a}')
```

```
Valor dentro de la función: 10
Valor fuera de la función: 5
```

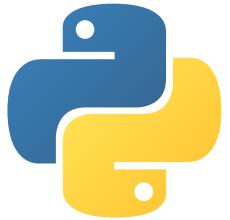


EJEMPLO MUTABLE

```
def modificar_lista(lista):
    lista.append(4)
    print(f'Lista dentro de la función: {lista}')

mi_lista = [1, 2, 3]
modificar_lista(mi_lista)
print(f'Lista fuera de la función: {mi_lista}')
```

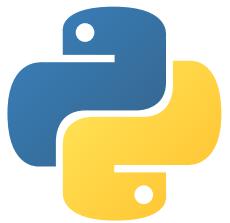
```
Listado dentro de la función: [1, 2, 3, 4]
Listado fuera de la función: [1, 2, 3, 4]
```



VARIABLES GLOBALES VS VARIABLES LOCALES

FUNCIONES EN PYTHON

- **VARIABLES LOCALES:** variables definidas dentro de la función. Tienen un ámbito local a la propia función. Por tanto, estos parámetros y variables no pueden ser utilizados fuera de la función porque no serían reconocidos.
- **VARIABLES GLOBALES:** definidas fuera de una función tienen un ámbito conocido como global y son visibles dentro de las funciones, donde solo se puede consultar su valor. Para modificar dentro de una función una variable definida fuera de la misma, hay que usar la palabra reservada **global**



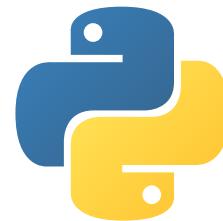
FUNCIONES EN PYTHON

ERROR
VARIABLES
LOCAL DE
FUNCIÓN
INVOCADA DESDE
FUERA

```
def saludo(nombre):
    numero = 10
    print("Hola", nombre)

#NameError: name 'numero' is not defined
print(numero)
```

```
", line 43, in <module>
    print(numero)
               ^
NameError: name 'numero' is not defined
```



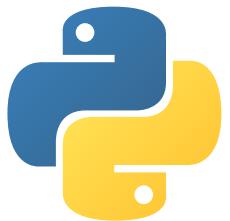
VARIABLE GLOBAL DESDE FUNCIÓN

FUNCIONES EN PYTHON

```
def verNumero():
    global numero
    numero=10
    print('numero: ', numero)

numero=20
verNumero()
print(numero)
```

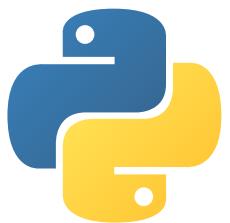
- Siempre se debe colocar la palabra global y el nombre de la variable global
- No se puede colocar la declaración y asignación en la misma.



CADENA DOCSTRING

FUNCIONES EN PYTHON

- La cadena docstring no es obligatoria pero sirve para documentar la función
- Debe especificar que hace y no como lo hace
- Va encerrada entre tres juegos de comillas dobles o simples
- Este texto aparece al escribir **help nombre_funcion**



FUNCIONES EN PYTHON

CADENA
DOCSTRING

```
def sumarValores(valor1, valor2):
    """Esta función recibe dos valores y devuelve la suma de ambos"""
    return valor1+valor2

help (sumarValores)
```

```
Help on function sumarValores in module __main__:

sumarValores(valor1, valor2)
    Esta función recibe dos valores y devuelve la suma de ambos
```

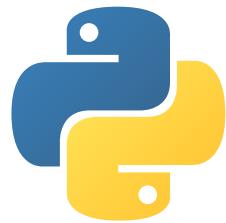
Práctica en clase

Realizar en Python

Desarrollar una función que reciba tres números enteros positivos y devuelva el mayor de los tres, sólo si éste es único (es decir el mayor estricto). Devolver -1 en caso de no haber ninguno.

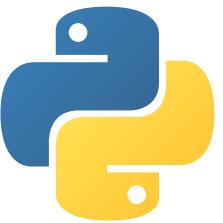
Desarrollar también un programa para ingresar los tres valores, invocar a la función y mostrar el máximo hallado, o un mensaje informativo si éste no existe.





FUNCIÓN CALCULARMAYOR

```
def calcularMayor(numero1,numero2,numero3):
    if numero1==numero2 or numero2==numero3 or numero3==numero1:
        return -1
    elif numero1>numero2 and numero1>numero3:
        return numero1
    elif numero2>numero1 and numero2>numero3:
        return numero2
    elif numero3>numero1 and numero3>numero2:
        return numero3
```



PROGRAMA PRINCIPAL

```
#Programa Principal

num1 = int(input("Ingrese el primer numero\n"))
num2 = int(input("Ingrese el segundo número\n"))
num3 = int(input("Ingrese el tercer numero\n"))

resultado= calcularMayor(num1,num2,num3)

if resultado!=-1:
    print("El mayor de los números es: ", resultado)
else:
    print("No hay un mayor estricto")
```

Práctica en clase

Realizar en Python

Una persona desea llevar el control de los gastos realizados al viajar en el subterráneo dentro de un mes.

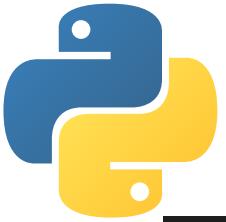
Sabiendo que dicho medio de transporte utiliza un esquema de tarifas decrecientes (detalladas en la tabla de abajo) se solicita desarrollar una función que reciba como parámetro la cantidad de viajes realizados en un determinado mes y devuelva el total gastado en viajes. .



Práctica en clase

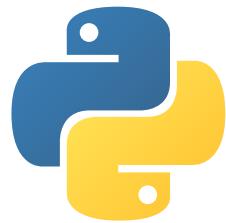
CANTIDAD DE VIAJES	VALOR DEL VIAJE
1 A 20	\$650
21 A 30	20% de descuento sobre tarifa
31 A 40	30% de descuento sobre tarifa
Más de 40	40% de descuento sobre tarifa





FUNCIÓN CALCULARCOSTOVIAJE

```
def calcularCostoViaje(cantViajes):
    if cantViajes<1:
        return -1
    elif cantViajes>=1 and cantViajes<=20:
        return cantViajes*650
    elif cantViajes>20 and cantViajes<=30:
        return cantViajes*(650*0.80)
    elif cantViajes>30 and cantViajes<=40:
        return cantViajes*(650*0.70)
    elif cantViajes>40:
        return cantViajes*(650*0.60)
```



PROGRAMA PRINCIPAL

```
#Programa Principal

cantidadViajes = int(input("Ingrese la cantidad de viajes que realizo en el mes\n"))

resultado=calcularCostoViaje(cantidadViajes)

if resultado!=-1:
    print(f'El costo total de sus viajes es: {resultado}')
else:
    print("La cantidad ingresada es invalida")
```

REPASO LISTAS





LISTAS

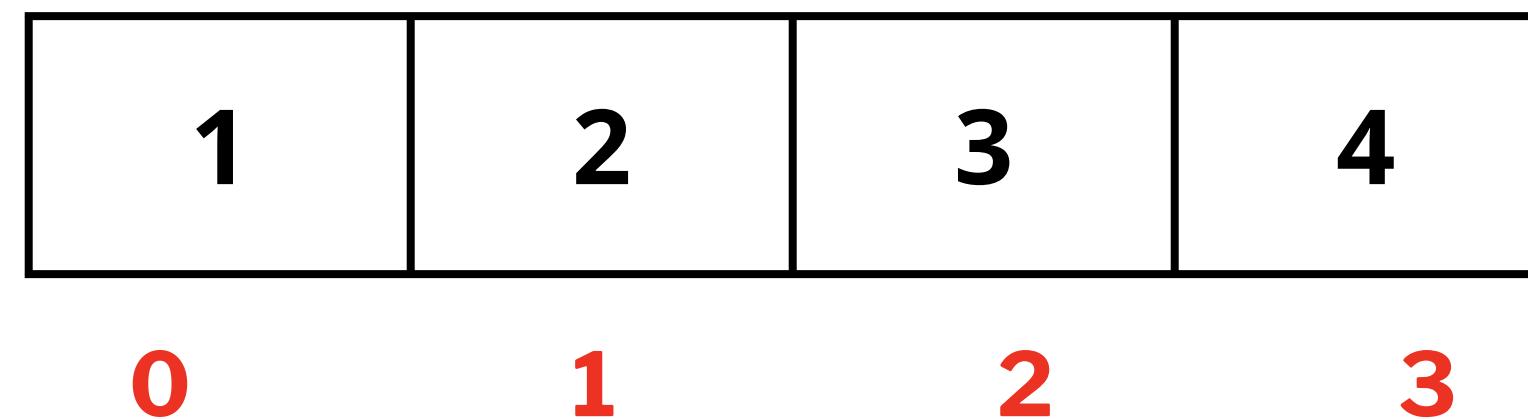
- Las listas son un tipo contenedor que se usan para **almacenar conjuntos de elementos relacionados del mismo o de distintos tipos.**
- Para crear una lista, simplemente hay que encerrar una secuencia de elementos separados por comas **entre corchetes []**.



LISTAS

- Crear una lista con números del 1 al 4 (mismo tipo de dato)

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





LISTAS

- Crear una lista con diferentes tipos de datos

```
lista2 = ["Pedro", 1.3, 56]
```

“Pedro”	1.3	56
---------	-----	----

0

1

2



LISTAS

- Acceder a elementos de una lista.
- Para acceder a un elemento de una lista se utilizan los índices. Un índice es un número entero que indica la posición de un elemento en una lista. **El primer elemento de una lista siempre comienza en el índice 0.**

```
print(lista2[0])  
print(lista2[1])  
print(lista2[2])
```

```
Pedro  
1.3  
56
```



LISTAS

Error al acceder a
un índice no existente

```
print(lista2[3])
~~~~~^~~
IndexError: list index out of range
```

Error al enviar como índice
un valor no numérico

```
print(lista2[1.4])
~~~~~^~~~~~^
TypeError: list indices must be integers or slices, not float
```



LISTAS

- Se puede añadir, eliminar o modificar elementos de una lista
- Para añadir un nuevo elemento se utiliza **append(elemento)**

```
vocales = ['a']

vocales.append('e')
vocales.append('i')

print(vocales)
```

```
[ 'a', 'e', 'i']
```



LISTAS

- **Empaquetado:** consiste en asignar un conjunto de variables o constantes a una lista

```
n1=10
```

```
n2=11
```

```
n3=14
```

```
lista=[n1,n2,n3]
```



LISTAS

- **Desempaquetado:** consiste en asignar los elementos de la lista a un conjunto de variables

```
dias= ["Lunes", "Martes", "Miércoles"]
```

```
variable1,variable2,variable3=dias
```



LISTAS

- **Concatenación de listas:** se pueden concatenar listas con el operador +

```
dias1= ["Lunes", "Martes", "Miercoles"]
```

```
dias2 =["Jueves", "Viernes", "Sabado", "Domingo"]
```

```
diasTotales = dias1+dias2
```

```
print(diasTotales)
```

```
['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo']
```



LISTAS

- **Replicar listas:** se pueden concatenar listas con el operador +

```
lista1 = [1,2,3]
```

```
lista1 = lista1*3
```

```
print(lista1)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```



LISTAS

- Para modificar un elemento de una lista, se utiliza el operador de **asignación "="**, que permite reemplazar el valor actual del elemento en la lista por otro valor nuevo.

```
vocales[0]='u' #Actualiza el indice cero con la letra u  
vocales[1]='o' # Actualiza el indice uno con la letra o
```



LISTAS

- Para eliminar un elemento de una lista, se puede realizar de varias formas.
- 1) Con la sentencia **del** se puede eliminar un elemento a partir de su índice:

```
vocales=['a','e','i','o','u']

print(vocales)

del vocales[0]

print(vocales)
```

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



LISTAS

2) También se puede usar los métodos remove().

Con remove() eliminamos la primera ocurrencia que se encuentre del elemento en una lista.

```
vocales=['a','e','i','o','u','i']

print(vocales)
vocales.remove('i')

print(vocales)
```

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



LISTAS

3) Con **pop([i])** obtiene el elemento **cuyo índice sea igual a i** y lo elimina de la lista.

Si no se especifica ningún índice, recupera y elimina el último elemento.

```
vocales=['a','e','i','o','u','i']

print(vocales)

vocales.pop(3)

print(vocales)
```

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



LISTAS

Eliminar todos los elementos de una lista, para esto se puede utilizar el metodo **clear()**

```
vocales=['a','e','i','o','u','i']
print(vocales)
vocales.clear()
print(vocales)
```

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



LISTAS

Para obtener la longitud de una lista, es decir que te devuelva la cantidad de elementos se utiliza el método **len**

```
vocales=['a','e','i','o','u','i']
print(len(vocales))
```

6



LISTAS

Tambien podemos utilizarlo para recorrer una lista.

```
vocales=['a','e','i','o','u','i']

for i in range (len(vocales)):
    print("Indice ", i,"Valor: ", vocales[i])
```

```
Indice 0 Valor: a
Indice 1 Valor: e
Indice 2 Valor: i
Indice 3 Valor: o
Indice 4 Valor: u
Indice 5 Valor: i
```



LISTAS

Operador **in** permite determinar si un elemento se encuentra dentro de una lista.

Este operador evalúa si el elemento en cuestión está presente en la lista especificada.

En caso afirmativo, devuelve True; de lo contrario, retorna False.

```
vocales=['a','e','i','o','u','i']

if 'a' in vocales:
    print("La letra a esta en la lista")
else:
    print("La letra a no se encuentra en la lista")
```

Práctica en clase

Realizar en Python

Leer un conjunto de números enteros, calcular su promedio e imprimir aquellos que superan el promedio.



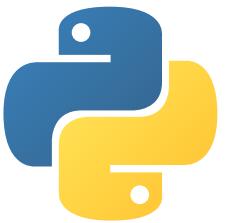
Práctica en clase

Realizar en Python

Desarrollar cada una de las siguientes funciones y escribir un programa que permita verificar su funcionamiento imprimiendo la lista luego de invocar a cada función:

- a. Cargar una lista con números al azar de cuatro dígitos. La cantidad de elementos también será un número al azar de dos dígitos.
- b. Calcular y devolver el producto de todos los elementos de la lista anterior.
- c. Eliminar todas las apariciones de un valor en la lista anterior. El valor a eliminar se ingresa desde el teclado y la función lo recibe como parámetro. No utilizar listas auxiliares.
- d. Determinar si el contenido de una lista cualquiera es capicúa, sin usar listas auxiliares. Un ejemplo de lista capicúa es [50, 17, 91, 17, 50].

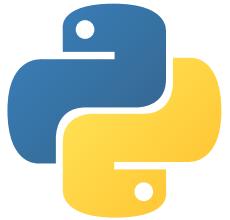




FUNCIÓN CARGARLISTA

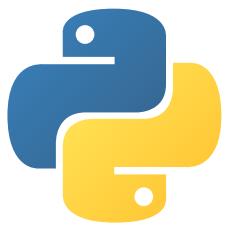
```
import random

def cargarLista():
    cantidadElementos = random.randint(10,99)
    lista=[]
    for i in range(cantidadElementos):
        lista.append(random.randint(1000,9999))
    return lista
```



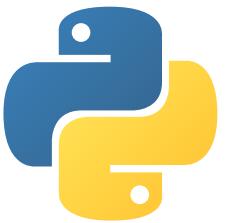
FUNCIÓN CALCULAR PRODUCTO

```
def calcularProducto(lista):
    resultado=1
    for i in range(len(lista)):
        resultado*=lista[i]
    return resultado
```



FUNCIÓN ELIMINARELEMENTO

```
def eliminarElemento(lista, valor):  
    for elemento in lista:  
        if elemento==valor:  
            lista.remove(valor)
```

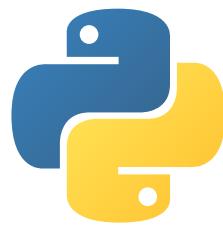


FUNCIÓN DETERMINARCAPICUA

```
def determinarCapicua(lista):
    izquierda = 0
    derecha = len(lista) - 1

    while izquierda < derecha:
        if lista[izquierda] != lista[derecha]:
            return False
        izquierda += 1
        derecha -= 1

    return True
```



PROGRAMA PRINCIPAL

```
#Programa Principal

print("Cargar lista - Punto 1\n")
lista=cargarLista()
print("Lista luego de cargar lista:",lista)

print("Producto: ", calcularProducto(lista))

elementoEliminar= int(input("Ingrese el número del elemento que desea eliminar\n"))
eliminarElemento(lista,elementoEliminar)
print("Lista resultante", lista)

print("La lista es capicua: ", determinarCapicua(lista))
```

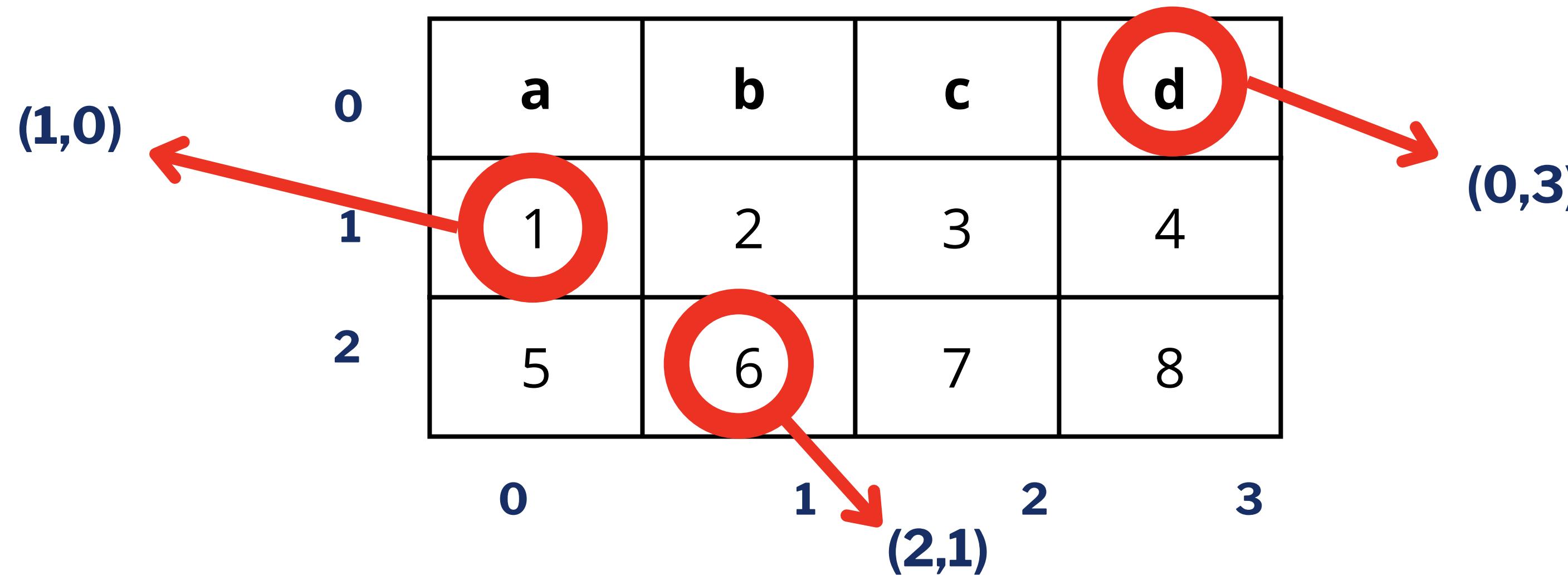
MATRICES



¿Que es una matriz?

- Es una estructura de datos formada por filas y columnas
- **Python** no tiene soporte para matrices de por sí, entonces se las simula construyendo **lista de listas**
- **Es decir una lista donde a su vez, sus elementos son listas**

Matrices



Matrices

- Creación de matrices

1. Forma estática:

```
matriz=[[0,0,0],  
        [0,0,0],  
        [0,0,0]]
```

```
print(matriz)
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Matrices

- Creación de matrices

2. Forma dinámica:

```
filas=3
columnas=4
matriz=[]

for f in range (filas):
    matriz.append([])
    for c in range(columnas):
        matriz[f].append(0)

print(matriz)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Matrices

- Creación de matrices

2. Forma dinámica utilizando el replicar:

```
filas=3
columnas=4
matriz=[]

for f in range (filas):
    matriz.append([0]*columnas)

print(matriz) | [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Recorrer una matriz

- Generalmente el recorrido de una matriz se realiza mediante el uso de un ciclo for anidado, que vaya accediendo uno por uno a los elementos que queremos acceder de la misma, tanto para la fila como para la columna.
- Al ser listas con listas en su interior, seguiremos teniendo acceso a las mismas funciones para **agregar, modificar y eliminar elementos como hemos visto anteriormente cuando trabajamos con listas de forma individual.**

Recorrer una matriz

```
for fila in range(0, len(matriz)):  
    for columna in range(0, len(matriz[fila])):  
        print("[", fila, "][", columna, "]", end=' ')  
        print("->", matriz[fila][columna])
```

[0][0] -> 0
[0][1] -> 0
[0][2] -> 0
[0][3] -> 0
[1][0] -> 0
[1][1] -> 0
[1][2] -> 0
[1][3] -> 0
[2][0] -> 0
[2][1] -> 0
[2][2] -> 0
[2][3] -> 0

Práctica en clase

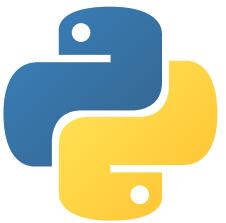
Realizar en Python

Desarrollar un programa que solicita la cantidad de alumnos de los que se desea cargar la información.

Luego por cada alumno se solicite el legajo y el nombre.

Imprima la matriz resultante



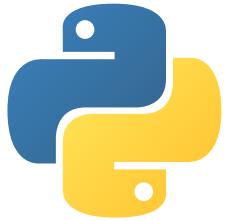


FUNCIÓN CARGARMATRIZ

```
def cargarMatriz(cantAlumnos):
    alumnos=[]

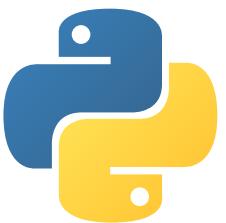
    for i in range(cantAlumnos):
        legajo = input("Ingrese el legajo del alumno: \n")
        nombre = input("Ingrese el nombre del alumno: \n")
        alumnos.append([legajo, nombre])

    return alumnos
```



FUNCIÓN IMPRIMIR MATRIZ

```
def imprimirAlumnos(alumnos):
    for i in range(len(alumnos)):
        print("Legajo: ",alumnos[i][0], "Nombre: ", alumnos[i][1])
```



PROGRAMA PRINCIPAL

```
#Programa Principal
cantAlumnos= int(input("Indique la cantidad de alumnos que desea inscribir\n"))

imprimirAlumnos(cargarMatriz(cantAlumnos))
```

Resumen de la clase

- Repaso de Funciones
- Repaso de Listas
- Matrices



EJERCITACIÓN FUNCIONES

Ejercitación

- **Ejercicio 1:** Desarrollar una función que reciba tres números enteros positivos correspondientes al día, mes, año de una fecha y verifique si corresponden a una fecha válida.

Debe tenerse en cuenta la cantidad de días de cada mes, incluyendo los años bisiestos. Devolver True o False según la fecha sea correcta o no. Realizar también un programa para verificar el comportamiento de la función.



Ejercitación

- **Ejercicio 2:** Un comercio de electrodomésticos necesita para su línea de cajas un programa que le indique al cajero el cambio que debe entregarle al cliente. Para eso se ingresan dos números enteros, correspondientes al total de la compra y al dinero recibido.

Informar cuántos billetes de cada denominación deben ser entregados como vuelto, de tal forma que se minimice la cantidad de billetes. Considerar que existen billetes de \$5000, \$1000, \$500, \$200, \$100, \$50 y \$10. Emitir un mensaje de error si el dinero recibido fuera insuficiente o si el cambio no pudiera entregarse debido a falta de billetes con denominaciones adecuadas. Ejemplo: Si la compra es de \$3170 y se abona con \$5000, el vuelto debe contener 1 billete de \$1000, 1 billete de \$500, 1 billete de \$200, 1 billete de \$100 y 3 billetes de \$10.



EJERCITACIÓN LISTAS

Ejercitación

- **Ejercicio 1:** Desarrollar cada una de las siguientes funciones y escribir un programa que permita verificar su funcionamiento imprimiendo la lista luego de invocar a cada función:
 - Cargar una lista con números al azar de cuatro dígitos. La cantidad de elementos también será un número al azar de dos dígitos.
 - Calcular y devolver el producto de todos los elementos de la lista anterior.
 - Eliminar todas las apariciones de un valor en la lista anterior. El valor a eliminar se ingresa desde el teclado y la función lo recibe como parámetro. No utilizar listas auxiliares.
 - Determinar si el contenido de una lista cualquiera es capicúa, sin usar listas auxiliares. Un ejemplo de lista capicúa es [50, 17, 91, 17, 50].



Ejercitación

- **Ejercicio 1:** Desarrollar cada una de las siguientes funciones y escribir un programa que permita verificar su funcionamiento imprimiendo la lista luego de invocar a cada función:
 - Cargar una lista con números al azar de cuatro dígitos. La cantidad de elementos también será un número al azar de dos dígitos.
 - Calcular y devolver el producto de todos los elementos de la lista anterior.
 - Eliminar todas las apariciones de un valor en la lista anterior. El valor a eliminar se ingresa desde el teclado y la función lo recibe como parámetro. No utilizar listas auxiliares.
 - Determinar si el contenido de una lista cualquiera es capicúa, sin usar listas auxiliares. Un ejemplo de lista capicúa es [50, 17, 91, 17, 50].



Muchas gracias!

Consultas?

