

# **Sesión 1:**

# **Fundamentos IA**

**Programa de inteligencia artificial**

**IBM SkillsBuild y SkillUp Online - Instructor Hugo Ramallo**



# Fundamentos de la IA

## DEFINICIÓN

Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones *comparables a las que realiza la mente humana*, como el aprendizaje o el razonamiento lógico. RAE



**ALAN TURING**  
**1950**

## ÁREAS DE LA IA



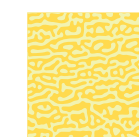
MACHINE LEARNING



NLP



SISTEMAS EXPERTOS



STT



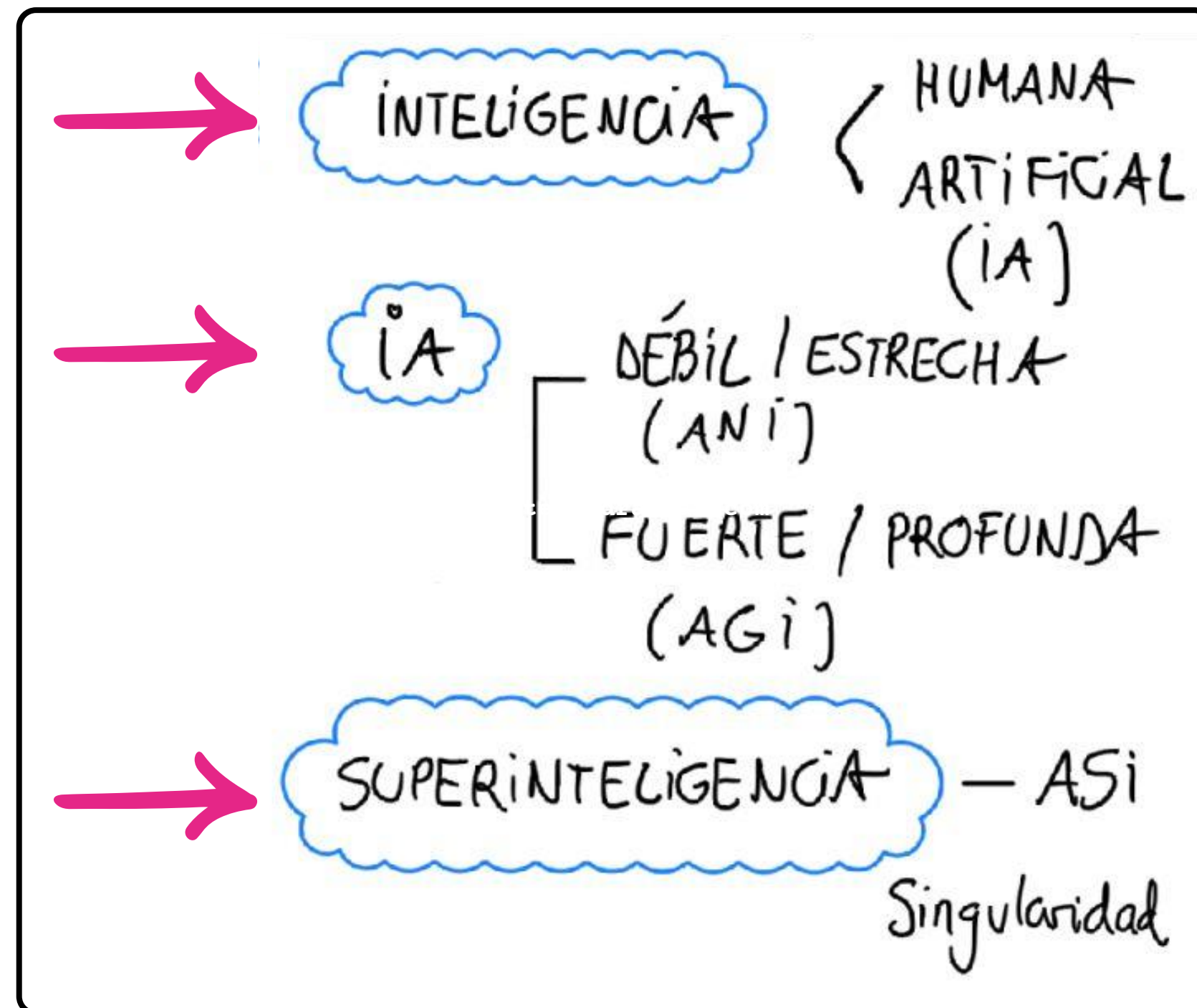
VISIÓN ARTIFICIAL



ROBÓTICA

# Fundamentos de la IA

## TIPOS DE IA

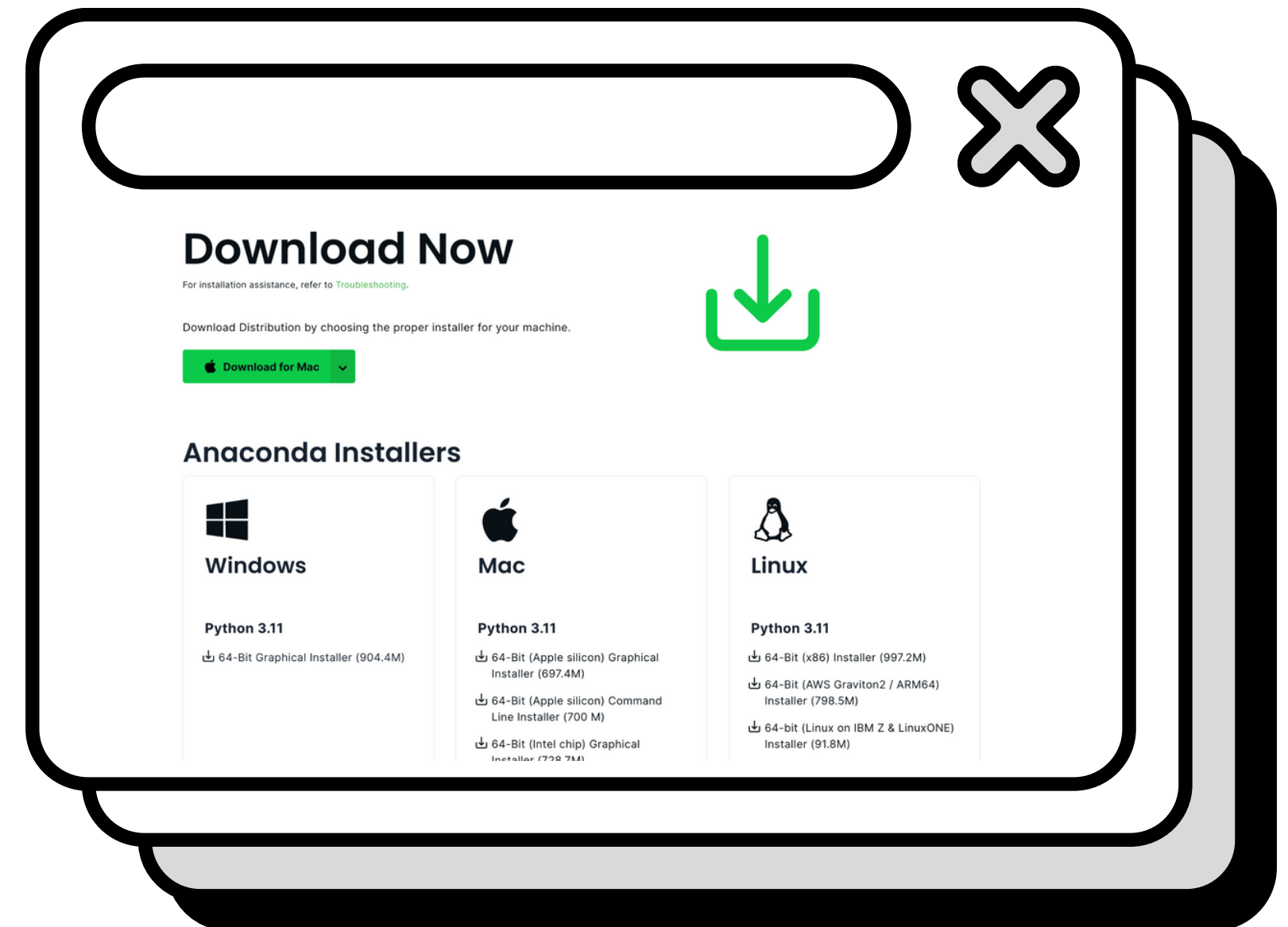
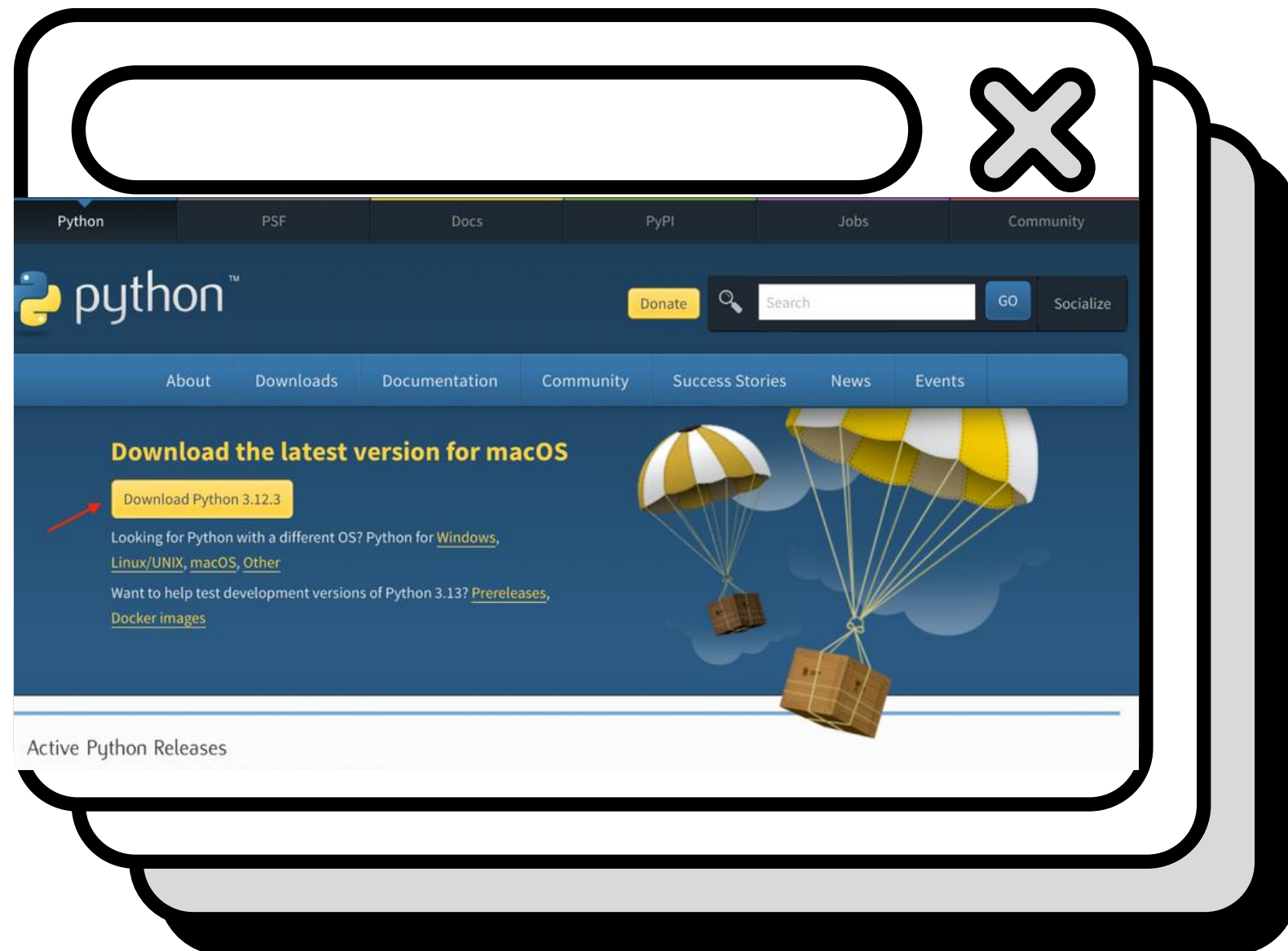


Fuente: <https://www.linkedin.com/in/doloresabuin/recent-activity/images/>



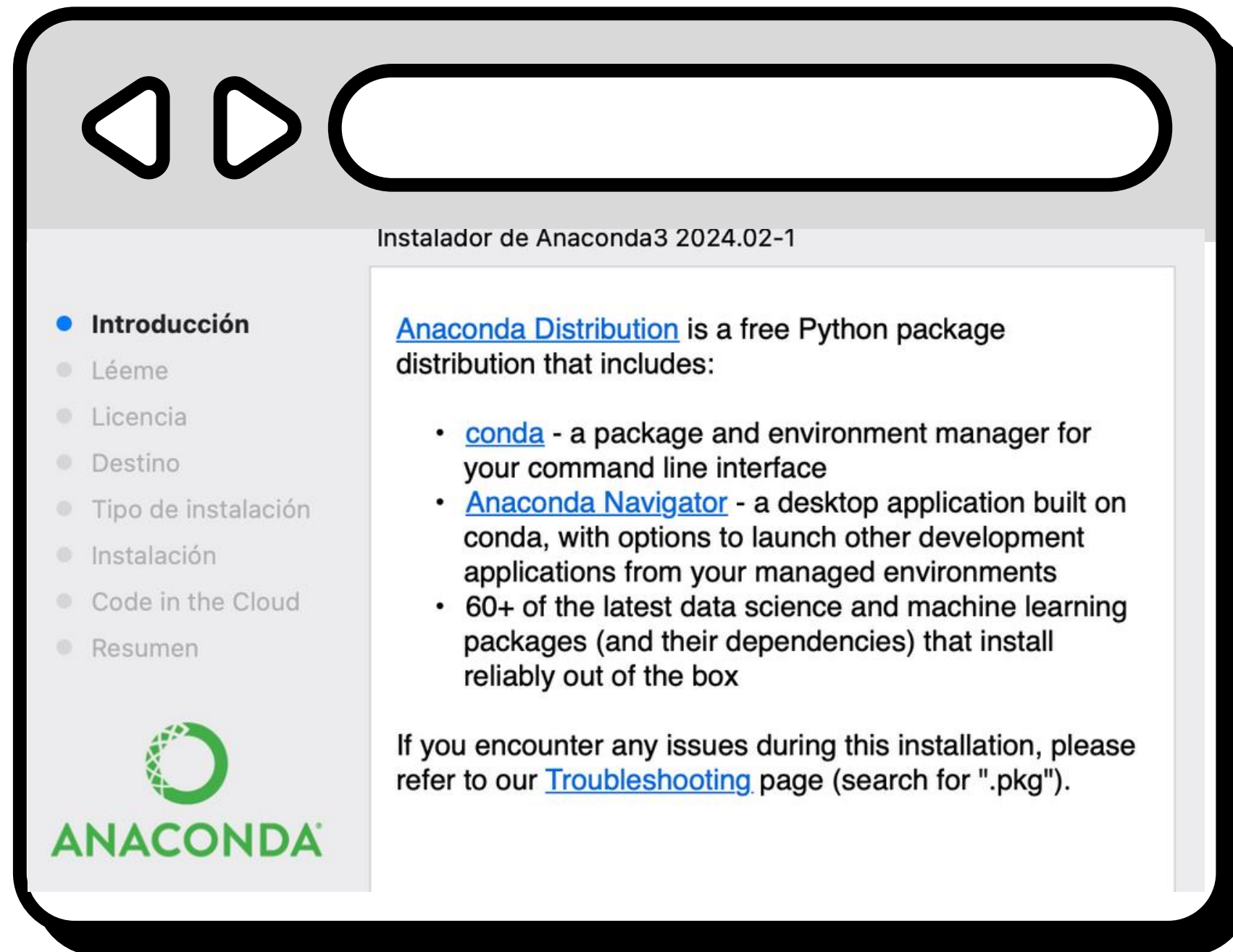
In collaboration with  
**IBM SkillsBuild**

# PYTHON – ANACONDA





# Instalación ANACONDA



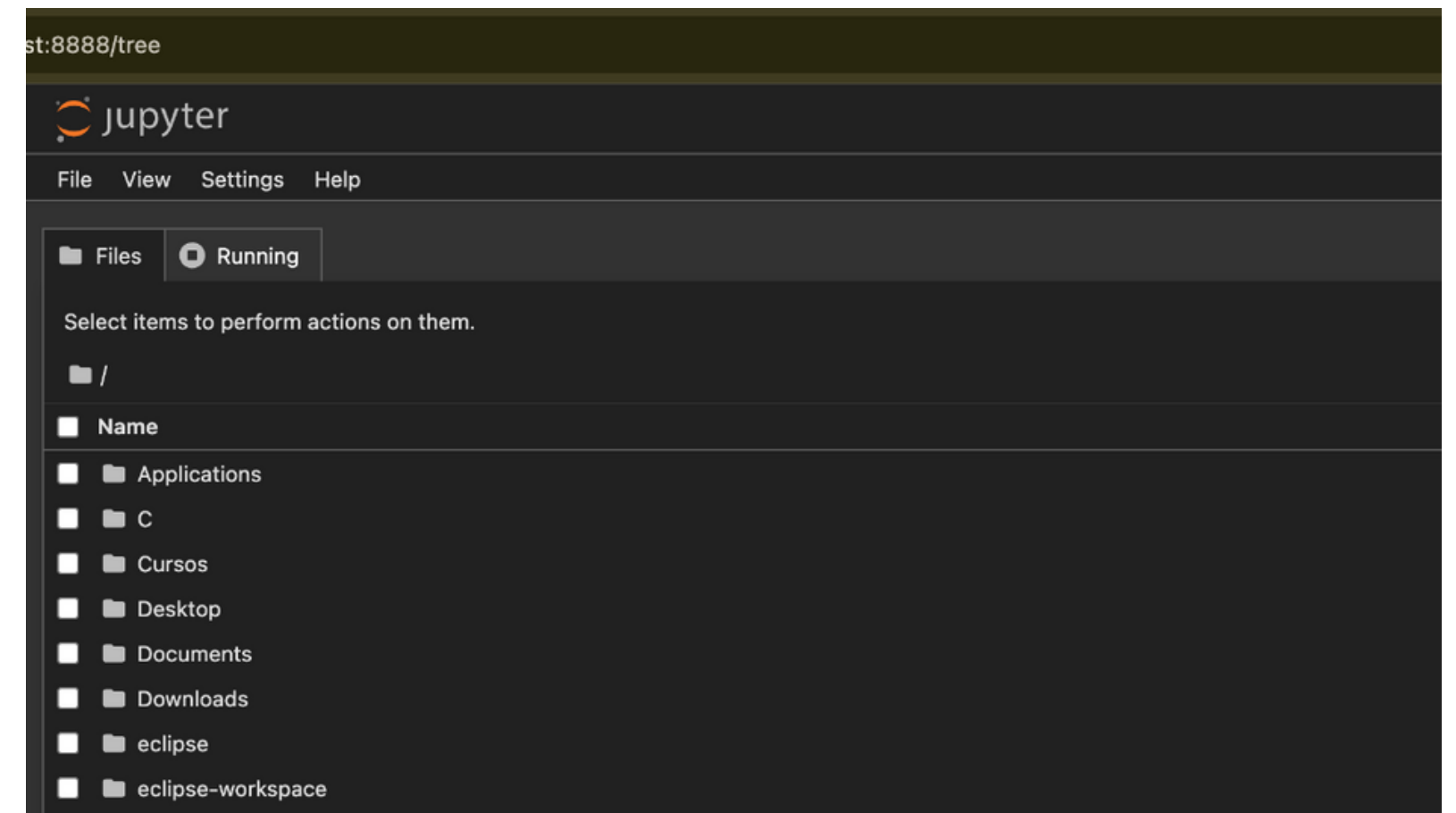
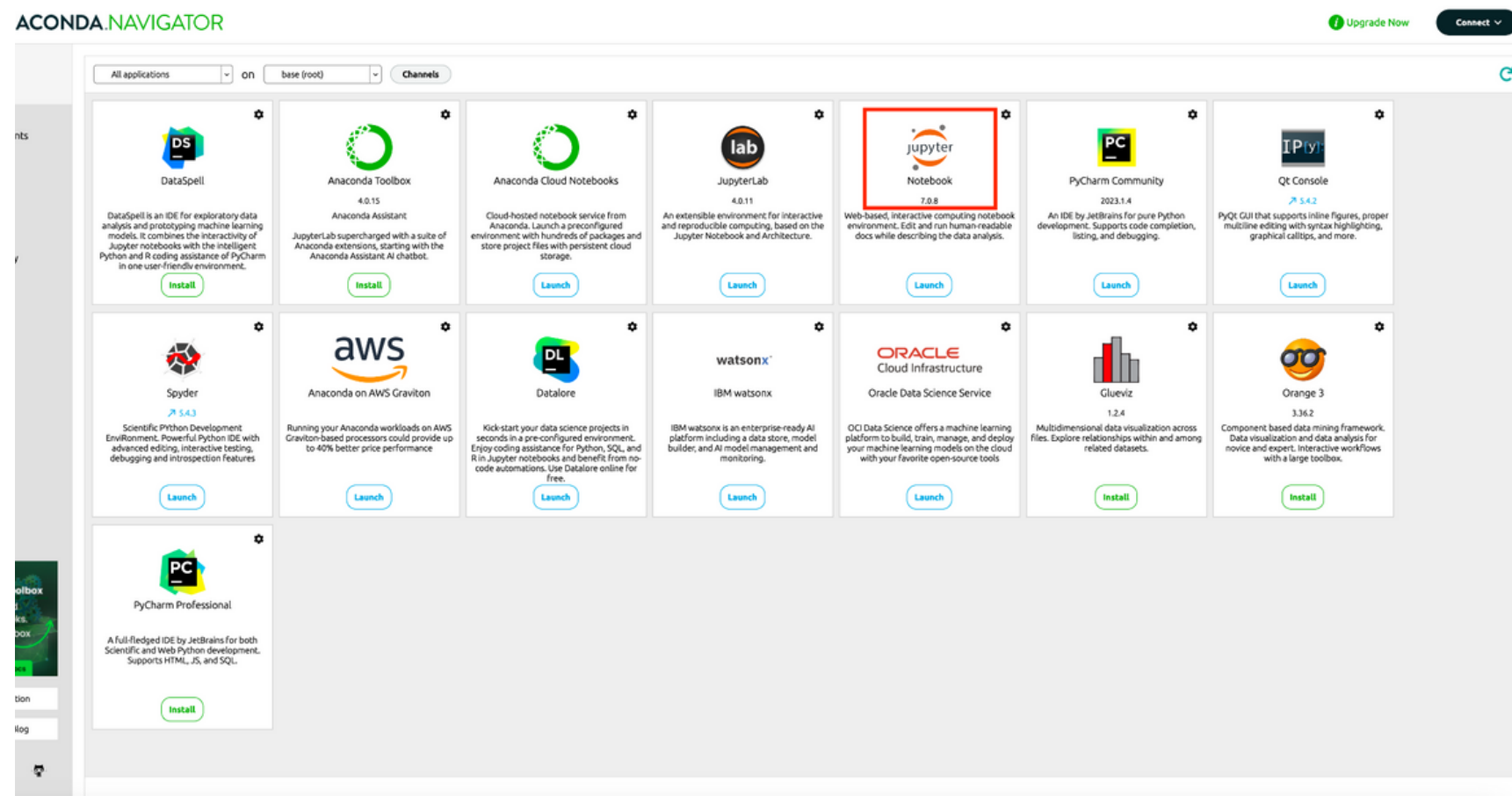
Una vez descargado el paquete Anaconda, el siguiente paso será proceder con la instalación.



# JUPYTER NOTEBOOK

● Al ejecutar el Anaconda-Navigator podremos ver la siguiente ventana

Ejecutar Jupyter Notebooks

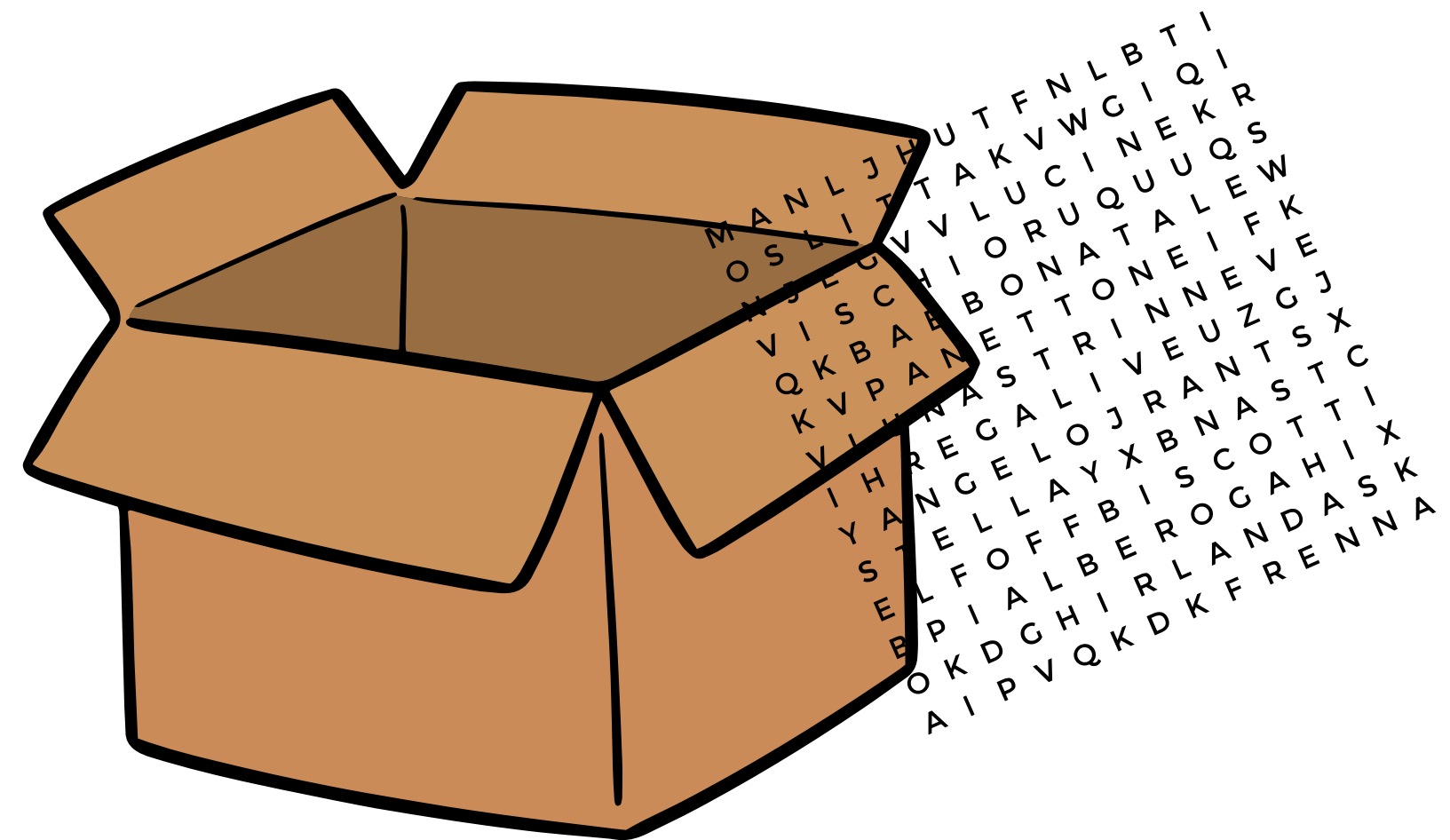


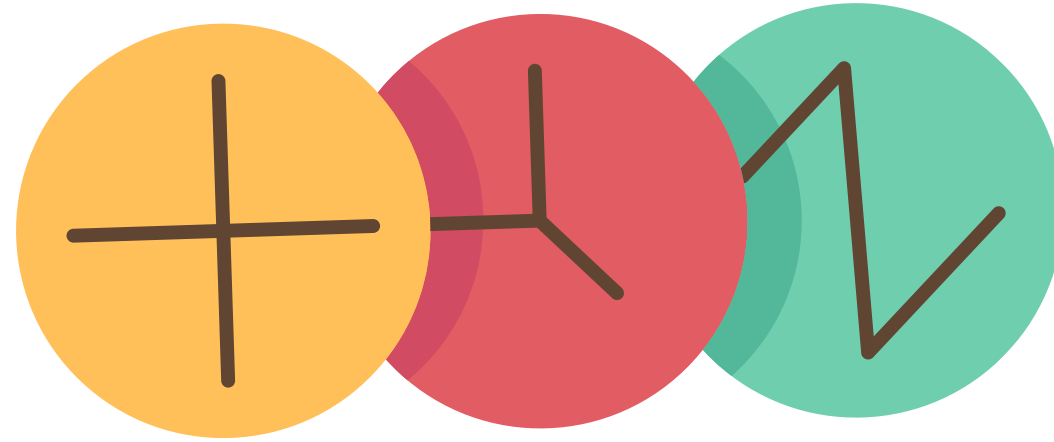


# VARIABLES Y TIPOS DE DATOS

Variables

Tipos de datos





## # Variables

Una variable en Python es un nombre que se asocia con un valor almacenado en la memoria. Las variables se utilizan para almacenar datos que pueden ser manipulados y utilizados a lo largo de un programa.

**A = 10**



# TIPOS DE DATOS

NÚMEROS (ENTEROS, REALES,  
COMPLEJOS)  
CADENAS  
BOOLEANOS  
\*LISTAS  
\*TUPLAS  
\*DICCIONARIOS  
SETS

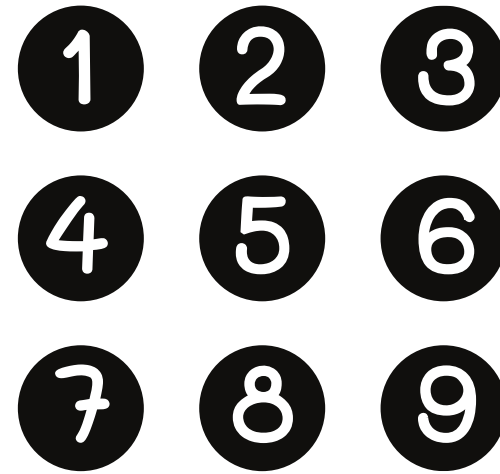




## # Tipos de datos

⇒ Los tipos de datos que se manejan en Python son básicamente: números (enteros, reales y complejos), strings (cadenas), booleanos, listas, tuplas, diccionarios, conjuntos(sets).

**En Python no hace falta definir el tipo de dato** (ya lo detecta Python), como si es necesario en otros lenguajes de programación, por ejemplo en Java, donde si hay que especificar si es un tipo entero (int), flotante (float), flotante extendido (double), booleano(boolean)



### # Enteros y flotantes

En Python, los números enteros (int) son valores sin decimales que pueden ser positivos, negativos o cero y tienen un rango prácticamente ilimitado. Los números reales (float) son valores con decimales que también pueden ser positivos, negativos o cero y se utilizan para representar números fraccionarios.

**Entero** = 5

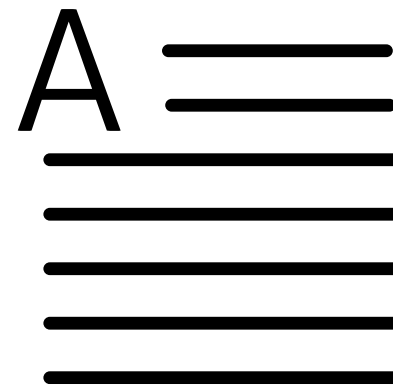
**Flotante** = 2,5

## # Cadenas

Una cadena, o string, en Python es una secuencia de caracteres encerrada entre comillas simples ( ' ') o comillas dobles ( " "). Las cadenas son utilizadas para almacenar y manipular texto.

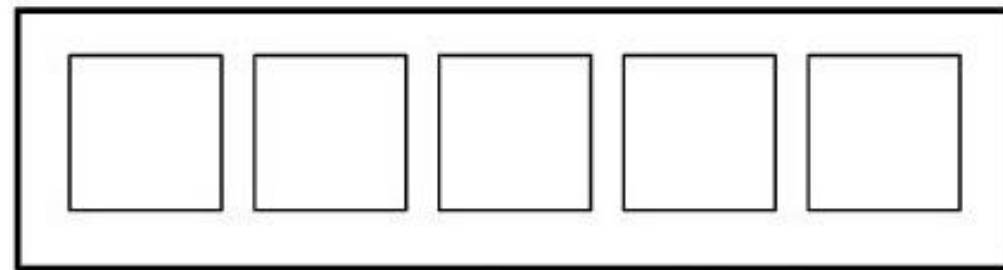
**"Esto es una cadena de texto"**

A



# PRÁCTICA 1: LISTAS

```
miLista = [1, 2, 4, 2.1 , "Hola"]
```



```
print( miLista[0] )
```

## FLEXIBLES Y MODIFICABLES

Se puede añadir, quitar y cambiar elementos.





## # Listas

Son secuencias mutables de elementos, que pueden ser de cualquier tipo y se pueden modificar después de su creación. Se usan normalmente para crear colecciones de datos. Se crean con corchetes [].

### Características Principales

- Ordenadas
- Mutables
- Permiten elementos duplicados
- Heterogéneas (pueden tener elementos de diferentes tipos)

### Ejemplos de uso

- Almacenar y manipular una colección de datos dinámicos.
- Iterar sobre elementos en bucles.
- Agregar, eliminar o modificar elementos de la colección.



**Lista** = [1, 2, 3, "a", "b", "c"]

# PRÁCTICA 1: TUPLAS

$T = (20, 'Juan', 35.75, [30, 60, 90])$

$T[0]$   $T[1]$   $T[2]$   $T[3]$

Ordenado: Mantiene el orden de la inserción de datos.

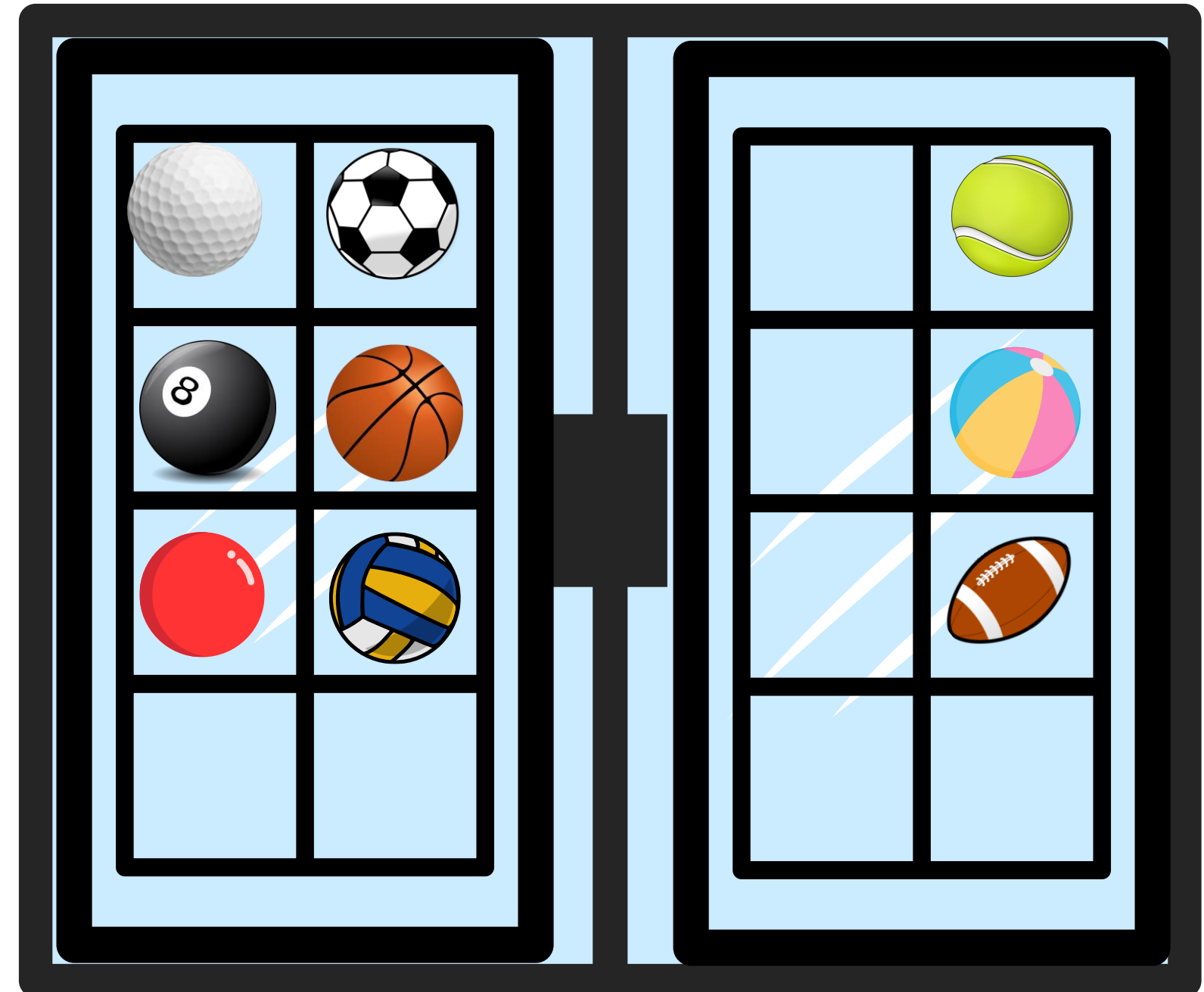
Inmutable: Las tuplas son inmutables y no podemos modificar los elementos.

Heterogéneo: Las tuplas pueden contener datos de diferentes tipos.

Contiene duplicados: Permite datos duplicados.

## FIJAS E INMUTABLES

No puedes cambiar, añadir ni quitar elementos.



## # Tuplas

Son secuencias inmutables de elementos, que también pueden ser de cualquier tipo, pero no se pueden modificar después de su creación.

### Características Principales

- Ordenadas
- Inmutables
- Permiten elementos duplicados
- Heterogéneas

### Ejemplos de uso:

- Almacenar datos constantes (días de la semana).
- Retornar múltiples valores desde una función.
- Utilizarlas como claves en un diccionario (debido a su inmutabilidad).

**Tupla** = (1, 2, 3, "a", "b", "c")



# PRÁCTICA 1: DICCIONARIOS

```
d = {'a': 10, 'b': 20, 'c': 30}
```

d['a'] d['b'] d['c']

**Desordenado:** Los elementos en el diccionario se almacenan sin ningún valor de índice.  
**Único:** Las claves en los diccionarios deben ser únicas.  
**Mutable:** Podemos agregar/modificar/eliminar pares clave-valor después de la creación.

## MODIFICABLES Y NO ORDENADOS

Un diccionario es una estructura de datos que almacena pares clave-valor

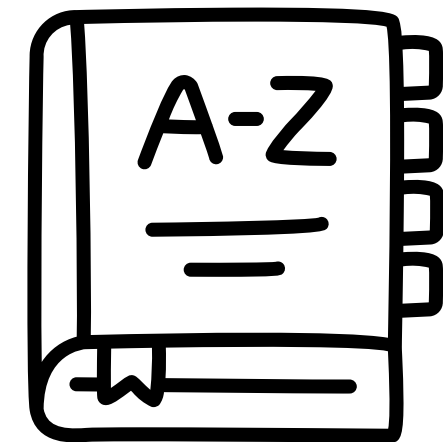


## # Diccionarios

Son colecciones desordenadas de pares clave-valor, donde cada clave única está asociada a un valor. Los diccionarios permiten un acceso rápido a los valores a través de sus claves. Se crean con las llaves {}.

### Características Principales

- Colección de pares clave-valor.
- Mutable (se pueden modificar).
- Ordenados a partir de Python 3.7
- Claves inmutables y únicas.



### Ejemplos de uso:

- Almacenar y acceder a datos estructurados.
- Implementar tablas de búsqueda y mapas.
- Organizar datos de una manera que permita el acceso rápido a través de claves.

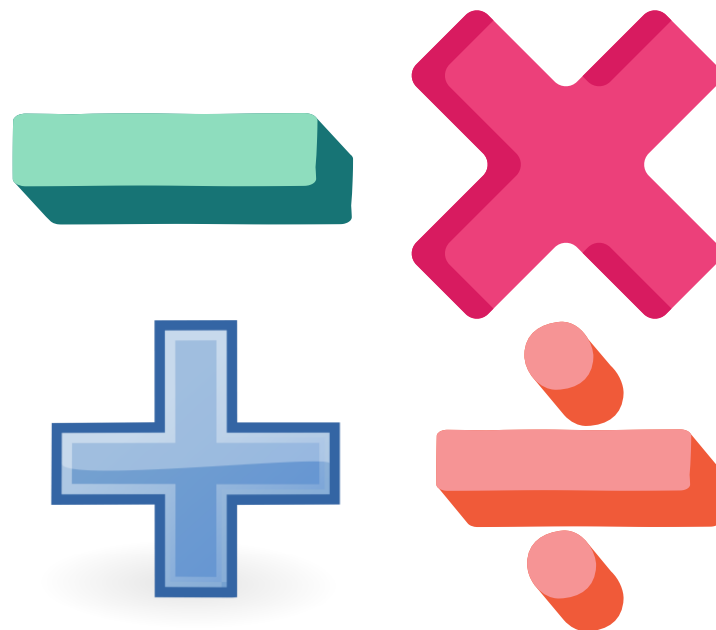
```
Diccionario = {"clave1": "valor1", "clave2":  
               "valor2"}
```





## PRÁCTICA 2: Comandos básicos

### OPERACIONES ARITMÉTICAS



#### Operadores Aritméticos En Python

	Operador	Significado	Ejemplo
1	+	Suma	$4 + 7 \rightarrow 11$
2	-	Resta	$12 - 5 \rightarrow 7$
3	*	Multiplicación	$6 * 6 \rightarrow 36$
4	/	División	$30 / 5 \rightarrow 6$
5	%	Módulo	$10 \% 4 \rightarrow 2$
6	//	Cociente	$18 // 5 \rightarrow 3$
7	**	Exponente	$3 ** 5 \rightarrow 243$

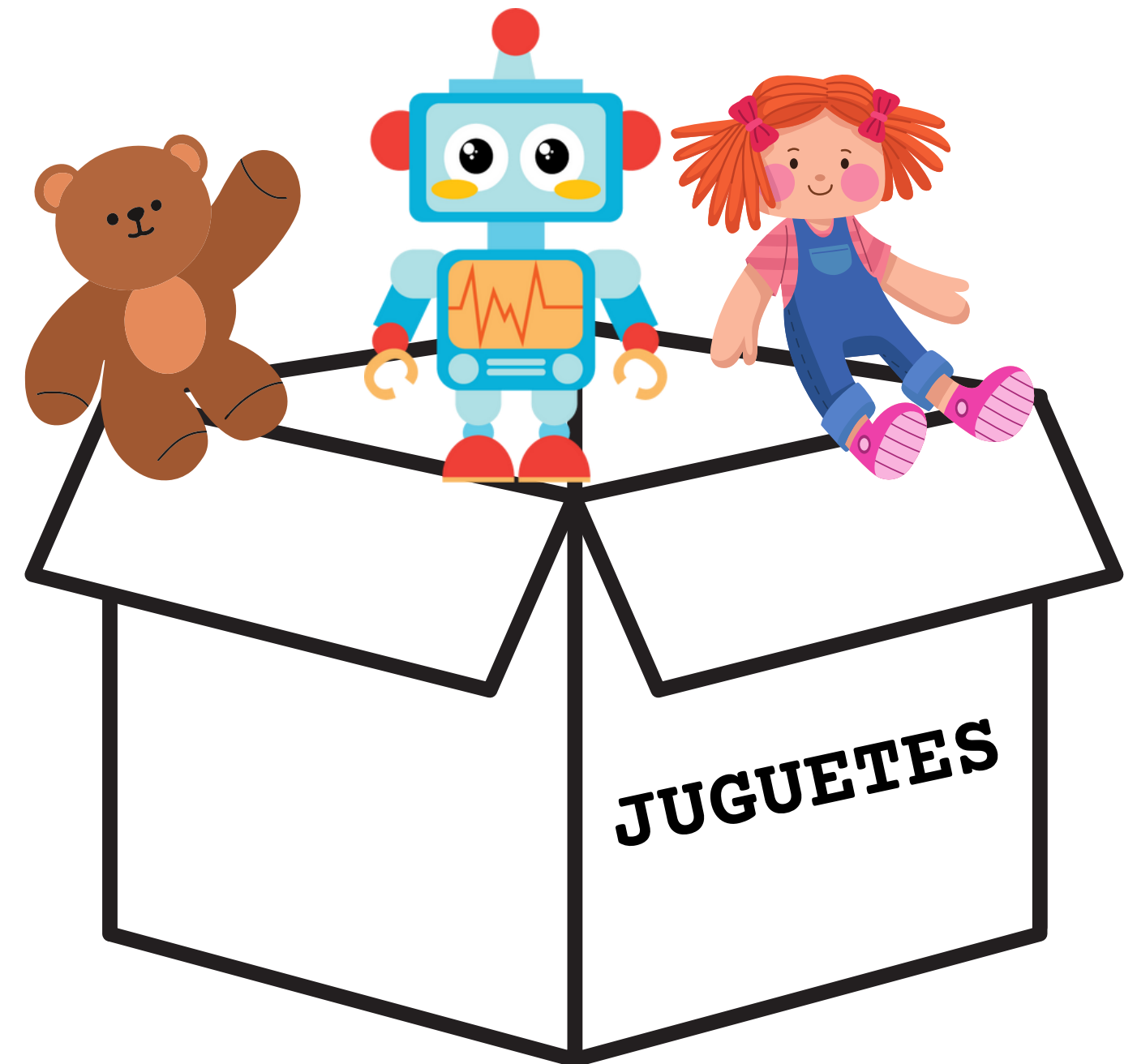


## PRÁCTICA 2: Comandos básicos

### ASIGNAR VARIABLES

Vincular un nombre (etiqueta) a un valor (contenido) utilizando el operador de asignación "="

```
# Asignar una variable en Python
juguetes = 3
print(juguetes) # Imprime 3
```





## PRÁCTICA 2: Comandos básicos

MANIPULAR SERIES



Biblioteca



Trabajar con una **colección de datos** ordenados para realizar operaciones como **añadir**, **cambiar**, **eliminar** y **analizar** los elementos.



## # Manipular series y asignar variables

Una Series en Pandas es una estructura de datos unidimensional similar a una columna en una tabla o una lista en Python, pero con funcionalidades adicionales. Las series son parte fundamental de la biblioteca pandas y están diseñadas para manejar y manipular datos etiquetados de manera eficiente.

### Características Principales

- Operaciones estadísticas
- Métodos y funciones integradas
- Eficientes

### Ejemplos de uso

- Análisis de precios de acciones
- Filtrado de temperaturas
- Análisis de series temporales





# PRÁCTICA 3: (for, while)

FOR

LUNES

MARTES

MIÉRC.

JUEVES

VIERNE  
S

SÁBADO

DOMING  
O

WHILE





### # Bucle For

El bucle for se utiliza para iterar (recorrer) sobre una secuencia (como una lista, una tupla, un diccionario, un conjunto o una cadena) y ejecutar un bloque de código para cada elemento de la secuencia.

#### Características

- Permite recorrer todos los elementos de una secuencia en el orden en que aparecen. Por ejemplo los días de la semana.

#### Casos de Uso

- Iterar sobre listas, tuplas, diccionarios, conjuntos y cadenas.
- Realizar operaciones repetitivas sobre elementos de una secuencia.

# FOR

```
frutas = ['manzana', 'banana', 'cereza']
for fruta in frutas:
    print(fruta)
```

### # Bucle While

El bucle while se utiliza para ejecutar un bloque de código repetidamente siempre que una condición booleana especificada sea True.

#### Características

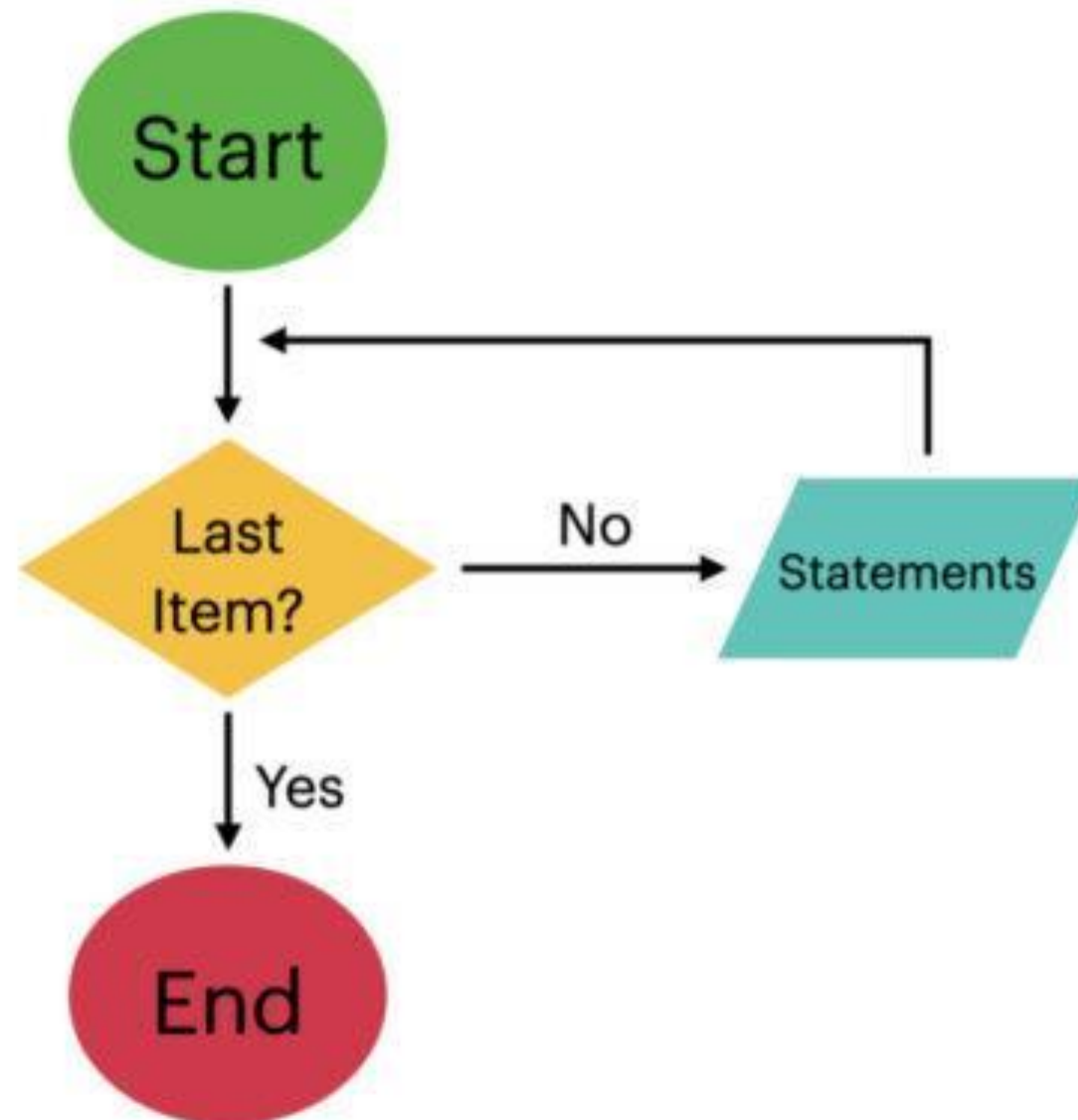
- Continúa ejecutándose mientras la condición especificada sea verdadera.
- Útil para situaciones donde el número de iteraciones no está definido de antemano.
- Si la condición nunca se vuelve falsa, el bucle puede continuar hasta el infinito.

#### Casos de Uso

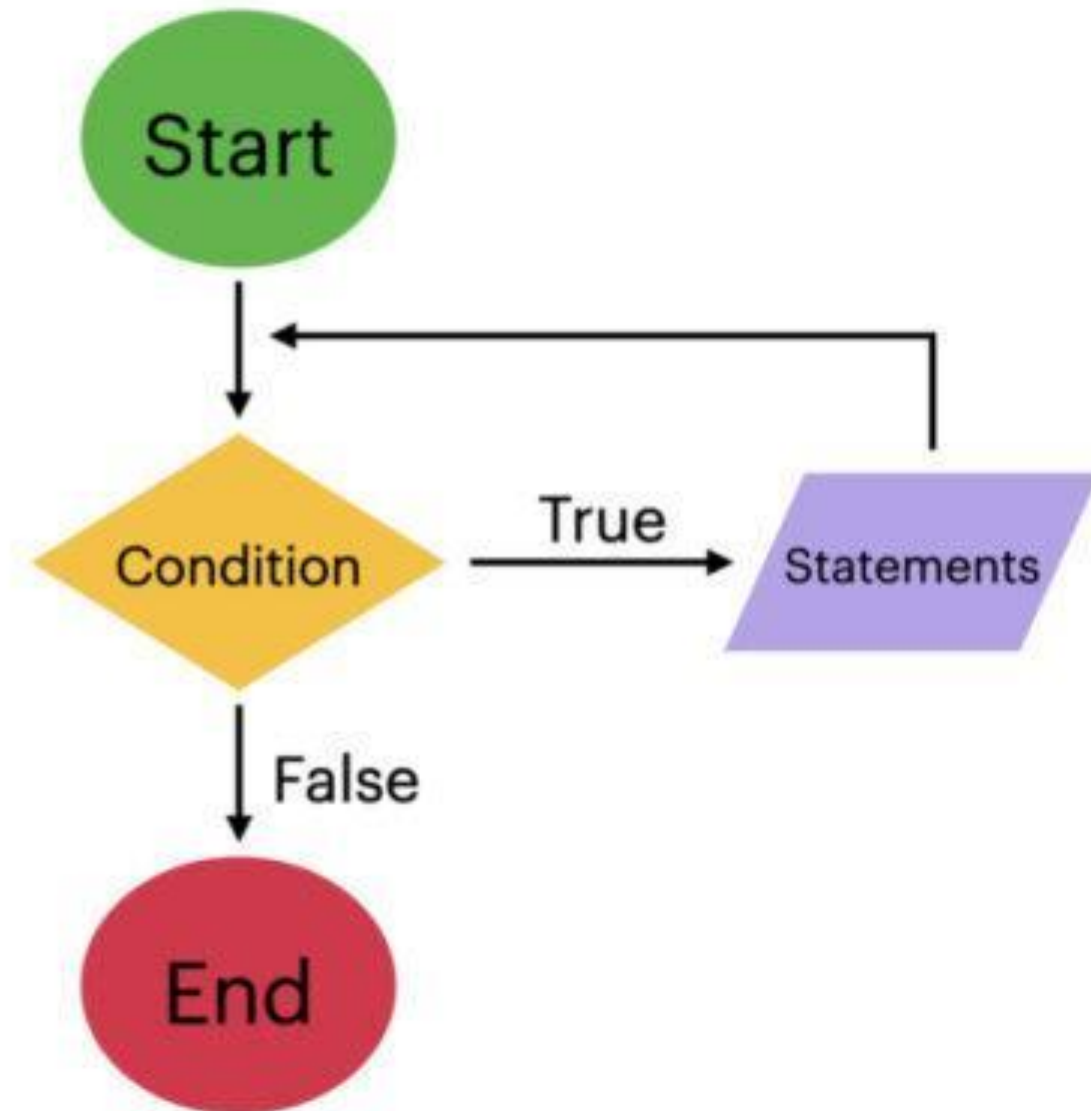
- Ejecutar un bloque de código mientras se cumpla una condición específica.
- Realizar operaciones hasta que se cumpla una condición de parada.

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

## For Loop



## While Loop



Fuente: edrawmax.com

## PRÁCTICA 3 : (IF-ELSE)

IF

ELSE

18+

```
edad = 18
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

## # IF - ELSE

La estructura de control if-else en Python se utiliza para tomar decisiones basadas en condiciones. Permite ejecutar un bloque de código si una condición es verdadera (True) y otro bloque de código si la condición es falsa (False).

### Características

- Ejecuta bloques de código basados en la evaluación de una condición booleana
- If-else puede ser anidada para evaluar múltiples condiciones.
- Puede incluir múltiples condiciones utilizando elif.

### Casos de Uso

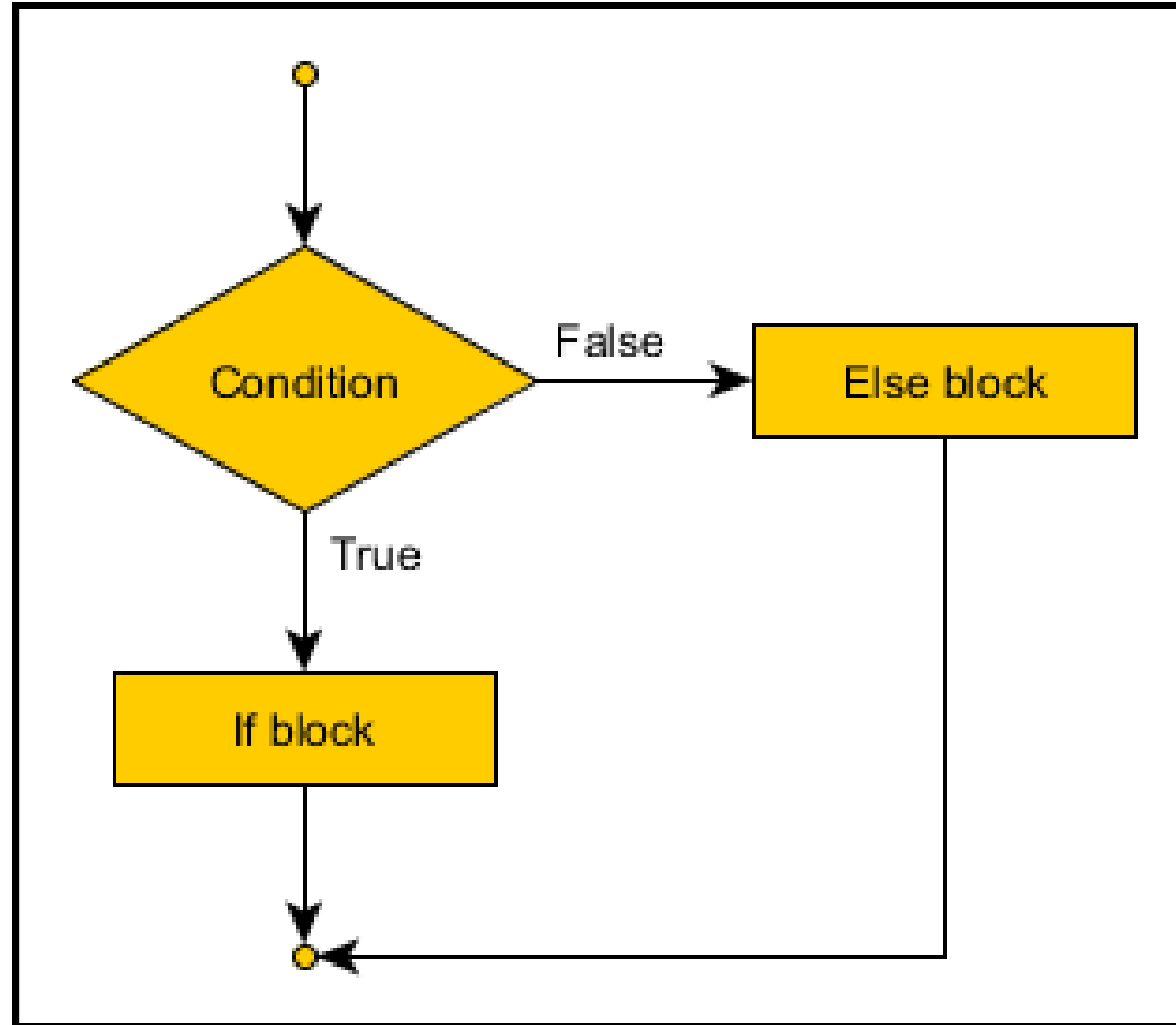
- Tomar decisiones en función de la entrada del usuario o de variables.
- Comprobar si los datos cumplen con ciertos criterios antes de procesar.

```
a = 10
b = 20
if a > b:
    print("a es mayor que b")
else:
    print("b es mayor que a")
```





# IF - ELSE Funcionamiento



# PRÁCTICA 4: TRY-EXCEPT

TRY

EXCEPT

```
# Ejemplo de manejo de división por cero
try:
    caramelos_por_amigo = 10 / 0 # Intentando dividir caramelos entre amigos
except ZeroDivisionError:
    print("¡Oops! No hay suficientes caramelos para todos los amigos.")
```



=  
0



## # Try-Except

La estructura de control try-except en Python se utiliza para manejar excepciones, es decir, errores que ocurren durante la ejecución de un programa. Permite que un programa continúe su ejecución incluso si se encuentra con un error.

### Características

- Captura y maneja errores que ocurren durante la ejecución del código dentro del bloque try.
- Opcionalmente, se pueden incluir bloques else y finally.

### Casos de Uso

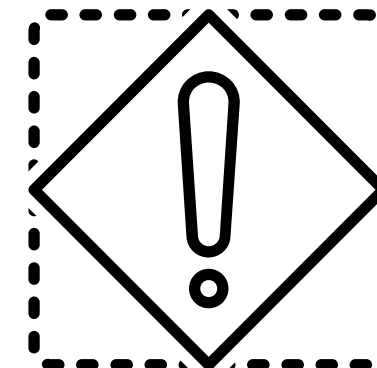
- Gestionar errores al leer o escribir archivos.
- Capturar y manejar entradas no válidas del usuario.
- Gestionar errores en conexiones de red o en operaciones de base de datos.

try:

```
    resultado = 10 / 0
```

except Exception as e:

```
    print(f"Ha ocurrido un error: {e}")
```



# ● ● ● PRÁCTICA 5 : FUNCIONES Y ENCAPSULAMIENTO DE BLOQUES

Los bloques de código, como instrucciones o funciones, se pueden agrupar y encapsular en secciones lógicas

```
# Definición de la función para contar juguetes
def contar_juguetes(caja_de_juguetes):
    total_juguetes = 0 # Inicializamos el contador de juguetes en cero
    for juguete in caja_de_juguetes: # Iteramos sobre cada juguete en la caja
        total_juguetes += 1 # Aumentamos el contador en uno por cada juguete
    return total_juguetes # Devolvemos el total de juguetes contados

# Lista de juguetes en la caja de tu amigo
caja_de_juguetes_de_tu_amigo = ["muñeca", "carro", "pelota", "rompecabezas"]

# Llamada a la función para contar los juguetes
total = contar_juguetes(caja_de_juguetes_de_tu_amigo)

# Mostramos el resultado
print("Tu amigo tiene", total, "juguetes en su caja.")
```





## # Funciones en Python

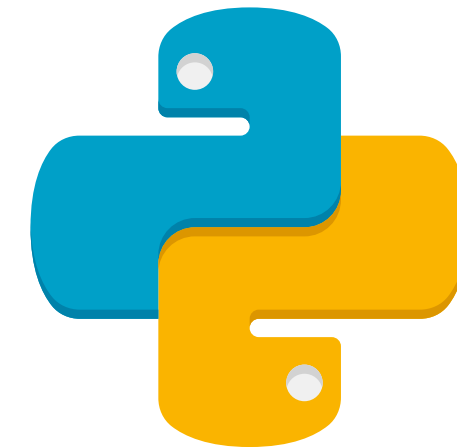
Una función en Python es un bloque de código reutilizable que realiza una tarea específica. Permite organizar el código en módulos más pequeños y manejables, facilitando la reutilización y la legibilidad.

### Características

- Agrupa varias instrucciones bajo un nombre, encapsulando la lógica para que se pueda reutilizar fácilmente.
- El código dentro de una función puede ser llamado múltiples veces en diferentes partes del programa, evitando duplicación.
- Permite dividir programas grandes en módulos más pequeños y manejables.
- Puede aceptar entradas (parámetros) para personalizar su comportamiento.
- Puede devolver resultados utilizando la palabra clave return.

```
def saludar():  
    print(";Hola, Mundo!")
```

```
saludar()
```





# **¡ Gracias !**

**Envíanos tus preguntas a  
[ibmskillsbuild.eu@skillup.online](mailto:ibmskillsbuild.eu@skillup.online)**