

# Introducción a Python

BOOTCAMP DATA ANALITICS & BIG DATA

Valencia, 17 de enero 2025

## Instructor:



Jorge Luzuriaga

✉ jluzuriaga@laberit.com

🌐 <https://www.linkedin.com/jluzuria2001>

## Sobre mi:

- Programador en **LÄBERIT**  
~ unidad DATOS
- Master (2012) & Doctor en informática (2017)  
por la  UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA
- Áreas de Interés:
  - ~ Redes Inalámbricas de Sensores
  - ~ Medidas de Prestación de la Red
  - ~ Conectividad y movilidad de dispositivos
  - ~ Internet de las Cosas
  - ~ Ciudades inteligentes
  - ~ Cambio climático

## Agenda

- Unidad 1: Introducción a Python
- Unidad 2: Python Básico
- Unidad 3: Configuración de Python para la ciencia de datos
- Unidad 4: Trabajar con datos
- Unidad 5: Visualizar información

## Requisitos

- Conocimientos básicos de programación
- Ordenador con instalado Python?
- Ganas de aprender

## Material del Curso

- <https://github.com/jluzuria2001/xxxxxxxxxxxxxxxxxx>
  - Presentación
  - Código de los ejercicios

**CampusDual TIC**

# **Unidad 1: Introducción a**

*Python*

## Introducción

- **Guido van Rossum** es el creador de Python
  - Actualmente ingeniero en Microsoft. ex-Dropbox, ex-Google, ...
- Bautizó el lenguaje con el nombre de Monty Python
- 1989: Creado en las Navidades (por aburrimiento)
- 1990: Lanzamiento
- 1995: ¿Python? ¿Qué es eso?
- 1997: ¡Pero si nadie usa Python!
- 1999: ¿Dónde puedo contratar programadores Python?
- 2000: Python 2.0 - uno de los lenguajes más populares
- 2008: Python 3.0 - cambios significativos



# Introducción

1. Python es un lenguaje de programación de **propósito general**
  - interpretado
  - interactivo
  - orientado a objetos
  - con semántica dinámica
2. Lenguaje **fácil y potente** (lo puede presumir)
3. **Sintaxis** elegante y **tipificación** dinámica
  - ideal para scripts y aplicaciones robustas
  - código sencillo de escribir y fácil de mantener
4. **Módular y paquetizado**
  - modularidad del programa y la reutilización del código
  - permite el desarrollo sea más rápido
5. **Distribución libre**
  - El intérprete, las bibliotecas (estándar y de terceros)
6. **Fácil de aprender**



## Material del Curso

- <https://github.com/jluzuria2001/xxxxxxxxxxxxxxxxxx>
  - Presentación
  - Código de los ejercicios

# ¿Por qué Python es tan popular?

1. Funciona en **todas las plataformas**
  - Windows, GNU/Linux, macOS
2. Es un **lenguaje polivalente** puede aplicarse en:
  - Automatización
  - Sitios/Aplicaciones Web
  - GUI
  - Servidores de red
  - APIs back-end
  - Aplicaciones de escritorio
  - Herramientas médicas
  - Dispositivos IoT
  - Analizar datos (big data)
  - Cálculo científico
  - Aprendizaje automático
3. Las **empresas tecnológicas** prefieren Python:  
Google, Youtube, Facebook, IBM,  
NASA, Dropbox, Yahoo, Mozilla,  
Quora, Instagram, Uber y Reddit, etc...

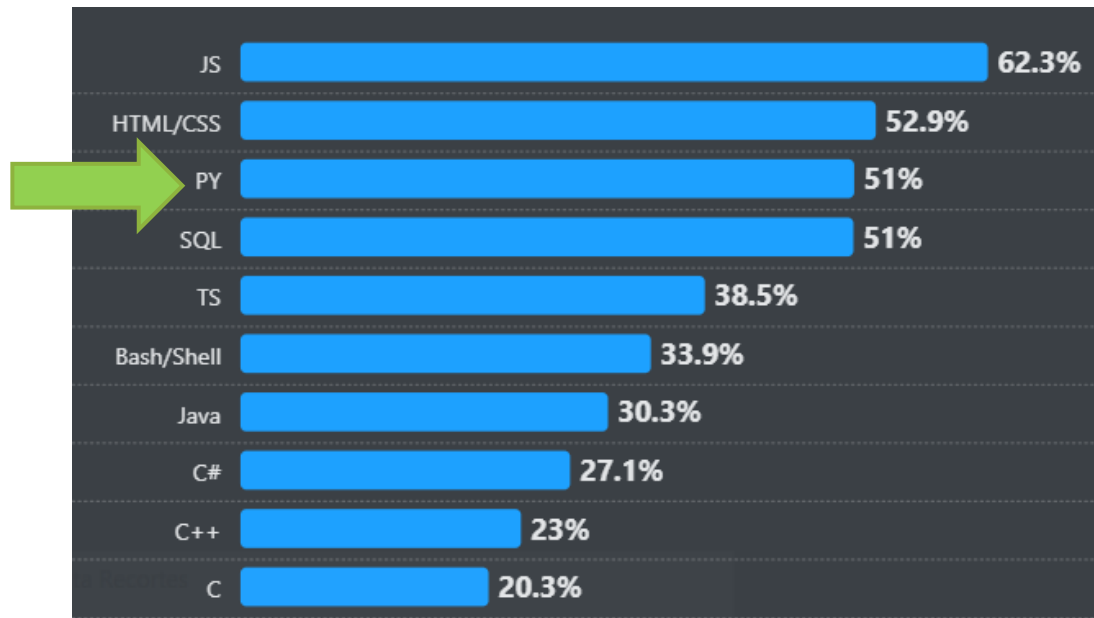
# Demanda de desarrolladores de Python

Tendencias (2024):

- Python es el segundo **lenguaje** de programación más **popular**
  - Tiene una curva de aprendizaje pronunciada
- Se utiliza para **construir soluciones digitales** basadas en:
  - Análisis de Datos (DA)
  - Aprendizaje Automático (ML)
  - Inteligencia Artificial (AI)
- Creciente popularidad de las tecnologías de IA, ML y DA
  - Continua demanda de programadores
  - Es una habilidad bien pagada

# Tecnologías más populares en el mundo

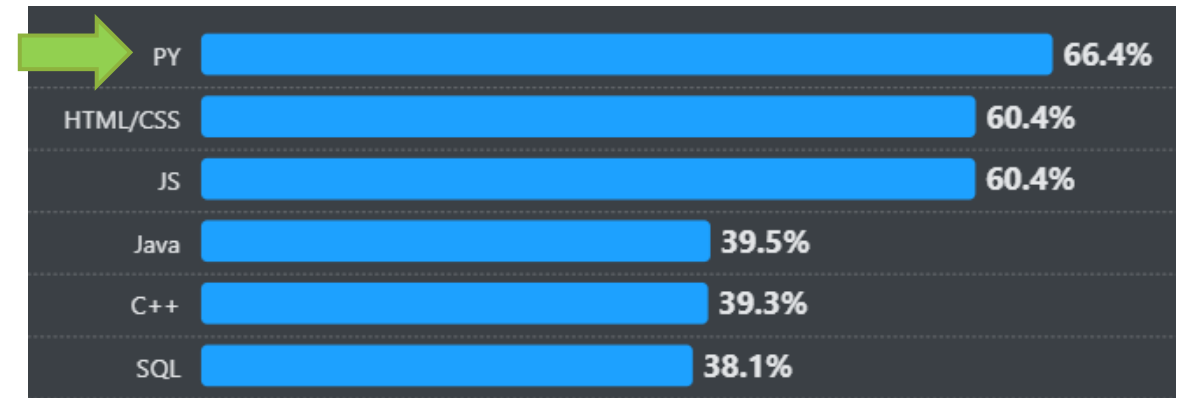
- 2024 – 60.171 encuestas \*



## Profesionales



## Estudiantes



\* <https://survey.stackoverflow.co/2024>

# ¿Qué hace un desarrollador de Python?

- Expresar y comunicar sus **ideas** a otros programadores (NO ordenadores)
- Es responsable de
  - **Escribir código** reutilizable y eficiente
  - Desarrollar la **lógica** de la aplicación (lado del servidor)
  - **Desplegar** elementos **de cara al usuario**

# Bibliotecas y paquetes de Python

- Alojadas en el repositorio de software (PyPI)  
“Índice de Paquetes de Python”



Hay más de  
600.544 proyectos  
892,601 usuarios  
y sigue creciendo ...

# Diferentes versiones de Python

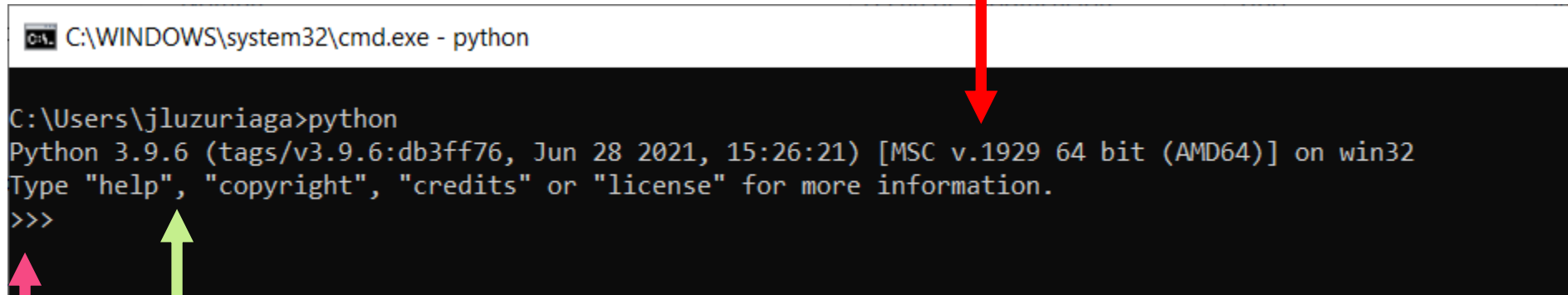
- Python 2 y Python 3 son las dos versiones populares
  - Ambas son muy similares
  - Algunas diferencias sintácticas
- Gran parte del código escrito para Python 3 se ejecuta en Py2 y viceversa
- **DEPRECATION**: *Python 2.7 reached the end of its life on January 1st, 2020.*
  - Python 3 es el único que está en desarrollo activo (en la actualidad)
    - Sólo se desarrolla para Python 3.
    - La mayoría de las librerías de terceros más utilizadas han sido portadas a Python 3
- Usaremos Python 3.8 para todos los ejemplos



# Comprobar la instalación de Python

versión principal y la fecha de lanzamiento

el sistema en el que se está ejecutando la shell interactiva



```
C:\WINDOWS\system32\cmd.exe - python

C:\Users\jluzuriaga>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

ejemplos de comandos que podemos escribir

el prompt >>> para introducir un comando



# Intérprete de Python

- Python cuenta con un IDLE (*Entorno integrado de desarrollo y aprendizaje*)
  - Una herramienta de desarrollo ideal para iniciarse
  - Permite probar programas fácilmente sin tiempo de espera
  - Se puede editar y crear fácilmente archivos con extensión `.py`.
- IDLE primero lee la línea de un programa
  - Evalúa sus instrucciones
  - Imprime en pantalla
  - Hace un bucle en la línea siguiente.
- IDLE proporciona tracebacks para mencionar errores

# ¿Cómo utilizar el IDLE de Python?

- La shell interactúa con el sistema operativo (Bash o CMD)
- Navegación en el shell
  - **Alt+P** recuperar una instrucción escrita previamente
  - **Alt+N** recuperar la siguiente instrucción
- Gestión y ejecución de los archivos de script
  - **Ctrl+N** nuevo
  - **Ctrl+O** abrir
- Escribir y formatear código

**CampusDual** TIC

# **Unidad 2:**

## **Python Básico**

# Objetivos de aprendizaje

- Utilizar Python IDLE (shell interactivo)
- Escribir y ejecutar scripts de Python
  - sencillos
  - dinámicos
- Utilizar variables y describir los diferentes tipos de valores que se pueden asignar
- Utilizar la entrada desde el teclado del usuario
- Entender la importancia de los comentarios y la indentación

## E1: Trabajar con el intérprete de Python (*shell*)

- Imprimir un mensaje
- Devuelve cualquier cosa que escribas
- Hacer operaciones aritméticas
- Salir

```
>>> print("¡Hola Mundo!")
```

```
>>> "Eco"  
"Eco"  
>>> 1234  
1234
```

```
>>> 9 + 3  
12  
>>> 100 * 5  
20  
>>> -6 + 5 * 3  
9
```

```
>>> exit()  
>>> quit()
```

## Actividad 1: Trabajar con el intérprete

1. Imprime el mensaje “esto es una cadena de texto”
2. Ejecuta la operación  $17 + 35 * 2$
3. Imprime los números del 1 al 9 en una sola fila (**sin bucles aún**)
4. Salir del interprete

Resultado de la ejecución las instrucciones

```
>>> ?  
esto es una cadena  
>>> ?  
87  
>>> ?  
1 2 3 4 5 6 7 8 9  
>>> ?
```

## Actividad 1: Trabajar con el intérprete

1. Imprime el mensaje “esto es una cadena de texto”
2. Ejecuta la operación  $17 + 35 * 2$
3. Imprime los números del 1 al 9 en una sola fila (**sin bucles aún**)
4. Salir del interprete

Resultado de la ejecución las instrucciones

```
>>> print("esto es una cade..")
esto es una cadena
>>> 17+35*2
87
>>> print("1 2 3 4 5 6 7 8 9")
1 2 3 4 5 6 7 8 9
>>> exit()
```

# Escribir y ejecutar scripts

- Shell
  - Probar una hipótesis rápida
  - Comprobar si un método específico existe para algún tipo de datos.
  - No se puede escribir un programa completo
- Script (modulo)
  - Cualquier cosa que se puede ejecutar en shell se puede ejecutar en un script
  - Un archivo que contiene instrucciones de Python
  - Debe tener la extensión de archivo **.py**



## E2: Crear un Script

1. Abre tu editor de texto
2. Crea un archivo llamado **ejercicio1.py** e inserta el comando para imprimir "hola mundo"
3. Guárdalo
4. Abre tu terminal (cambia al directorio donde se encuentra el archivo)
5. Ejecuta:  
python **ejercicio1.py**.

# Ejecución de un archivo que contiene comandos no válidos

- La introducción de instrucciones no válidas provoca un error
- En una salida que se llama **trazas** (**stack trace**)
- Nos dice cosas útiles del error como:
  - **dónde** ocurrió
  - **tipo**
    - qué otras llamadas se dispararon en el camino
- Las trazas deben leerse de **abajo a arriba**
- Identificar lo qué está causando el error para actuar en consecuencia

## E3: Pasar argumentos de usuario a los scripts

- Scripts dinámicos
    - Hacer que el usuario proporcione argumentos al llamar al script
1. Crea un nuevo archivo llamado `ejercicio2.py`
  2. Añade el siguiente código:

```
import sys
sys.argv[1]
```
  3. Guarda y ejecuta el script pasándole un argumento  
`python ejercicio2.py argumento1`

## Actividad 2: Ejecutar scripts simples

- Crear un script para generar de tarjetas de presentación
- Se deben pasar dos parámetros con el script (nombre y apellido)

1. Crear un script llamado **tarjeta.py**
2. Utilizar la sentencia para imprimir
  - Nombre y Apellido
  - 20 guiones bajos (como bordes superior e inferior)
3. Ejecute el script pasando dos argumentos

```
-----  
Primer nombre: jorge  
Segundo nombre: Luzuriaga  
-----
```

```
python tarjeta.py jorge Luzuriaga
```

# Sintaxis de Python

- Lo que se necesita para escribir un programa
- Cómo se estructuran las expresiones del lenguaje
- Variables
  - Son referencias a valores en memoria
  - Usadas para almacenar cualquier tipo de datos
  - El valor y el tipo de una variable pueden cambiar durante el tiempo de ejecución
- Valores
  - Pueden se asignados a las variables
  - **Tipos:** cadenas, enteros, decimales (flotantes), booleanos, y estructura de datos
  - Comprobación del tipo de un valor

```
>>> type(valor)
```

# Tipos de valores

- Valores numéricos - Enteros
  - Cualquier número positivo, negativo, incluyendo el 0
  - Nunca puede ser una fracción, un decimal, o un porcentaje

```
>>> type(3)  
<class 'int'>
```

- Cadena
  - Secuencia de caracteres que se coloca entre dos comillas (simples o dobles)

```
>>> type('esto es otra cadena')  
<class 'str'>
```

## Conversión de tipos

- Una cadena con un valor entero dentro

```
>>> int(valor_cadena)
```

- Un entero se quiere poner en una cadena

```
>>> str(valor_entero)
```

## Uso de variables

- Utilizamos las variables cuando en nuestro código tenemos un valor que queremos utilizar varias veces

```
>>> numero = 7
>>> numero * 5
>>> numero + 2
>>> numero / 3,5
>>> numero - numero
```

El valor de numero no cambiará a menos que lo reasignemos



## Uso de variables

- Utilizamos las variables cuando en nuestro código tenemos un valor que queremos utilizar varias veces

```
>>> numero = 7
>>> numero * 5
>>> numero + 2
>>> numero / 3,5
>>> numero - numero
```

El valor de numero no cambiará a menos que lo reasignemos

```
>>> print(numero)
7
>>> numero = 22
>>> print(numero)
22
```

## Uso de variables

- Utilizamos las variables cuando en nuestro código tenemos un valor que queremos utilizar varias veces

```
>>> numero = 7
>>> numero * 5
>>> numero + 2
>>> numero / 3,5
>>> numero - numero
```

El valor de numero no cambiará a menos que lo reasignemos

```
>>> print(numero)
7
>>> numero = 22
>>> print(numero)
22
```

```
>>> x = numero +
1
>>> x
```

# Uso de variables

- Utilizamos las variables cuando en nuestro código tenemos un valor que queremos utilizar varias veces

```
>>> numero = 7
>>> numero * 5
>>> numero + 2
>>> numero / 3,5
>>> numero - numero
```

El valor de numero no cambiará a menos que lo reasignemos

```
>>> print(numero)
7
>>> numero = 22
>>> print(numero)
22
```

```
>>> x = numero +
1
>>> x
```

```
>>> mensaje = "I love
Python"
>>> mensaje + "!" * 3
```

# Asignación múltiple

- Asignación de múltiples variables en una sentencia

```
>>> a, b, c = 1, 2, 3
>>> print(a)
1
>>> print(b)
2
>>> print(c)
3
```

## Actividad 3: Uso de variables y sentencias de asignación

- Escribir un **script para calcular la velocidad**
- utilizando la distancia (kilómetros) y el tiempo (horas)
- Se sabe que:
  - La fórmula para calcular la velocidad es distancia/tiempo

$$\bar{v} = \frac{d}{t}$$

- Probar asignando:
  - Distancia = 150 y tiempo = 2

**La velocidad en kilómetros por hora es: 75.0**

## Actividad 3.1: Uso de variables y sentencias de asignación

- Modificar el **script para calcular la velocidad** creado en la actividad anterior para transformar la velocidad en función de:
  - millas por hora
  - metros por segundo
- Se sabe que para convertir
  - Los kilómetros en millas: se divide los kilómetros por 1,6
  - Los kilómetros en metros: se multiplica los kilómetros por 1.000
  - Las horas en segundos: se multiplica las horas por 3.600
- Probar asignando las entradas del usuario probar con :
  - Distancia = 150 y tiempo = 2

La velocidad en kilómetros por hora es: 75.0

La velocidad en millas por hora es: 46.875

La velocidad en metros por segundo es: 20.833333333333332

# Reglas para nombrar variables

1. Puede tener letras, símbolos y dígitos
2. No puede comenzar con un dígito

**7x** es un nombre de variable **inválido**

3. Debe evitarse los espacios
4. No se puede utilizar palabras clave de Python

*import, if, for, lambda, ...*

Ninguna palabra del listado que se muestra en :

```
help()
help> keywords
```

```
ctrl+c (para salir)
```

# Convenciones de nomenclatura de Python

- **Nombres de variables compuestos** **deben** escribirse en notación snake\_case

```
>>> primera_letra = "a"      # Snake
>>> primeraLetra = "a"      # Camel
>>> PrimeraLetra = "a"      # Pascal
```

- **Nombres de las constantes** **deben** escribirse en mayúsculas para denotar que sus valores no deben cambiar

```
>>> NUMERO_DE_PLANETAS = 8
```

- Evitar nombres de variables de un solo carácter que pueden confundirse ejemplo las mayúsculas I y O con 1 y 0



## Actividad 4: Asignación de variables

- Escribir un script que calcule el área y la circunferencia de un círculo con un radio 7.

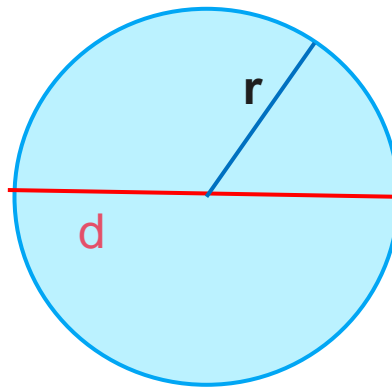
- declara el valor constante de  $\pi$  (PI) = 3,14159

- Se sabe las fórmulas para calcular

Área  $A = \pi r$

Circunferencia  $C = 2\pi r$

Diámetro  $d = 2r$



- Resultado de la ejecución del script

Área: 153.93791

Circunferencia: 43.98226

# Entrada de usuario - función input()

- Para obtener la entrada del teclado del usuario desde el CLI
  - Se detiene la ejecución del programa hasta que el usuario después de completar su entrada presiona la tecla *Enter*
- No hay ninguna señal que nos permita saber cuándo escribir
  - Se puede pasar un aviso a la función de entrada

```
>>> mensaje = input("Introduce un texto: ")
```

- Los valores devueltos por la función de entrada son siempre cadenas



# #Comentarios

- Son una parte fundamental en programación
- Hacen que el código sea más fácil de entender para los humanos
- Puede decirnos
  - Por qué se tomaron ciertas decisiones
  - Mejoras que se pueden hacer en el futuro
  - Explicar la lógica del negocio
- Hay tres formas de escribir comentarios
  1. Cadenas de documentación (docstrings) (envueltas entre comillas triples `'''`)
  2. Comentarios en línea (comienza con un signo #, misma línea)
  3. Comentarios en bloque (comienza con un signo #, mismo nivel de sangría)

# #Comentarios

1. Cadenas de documentación (docstrings) (envueltas entre comillas triples `'''`)

```
'''  
Esta función toma dos números como entrada  
'''
```

2. Comentarios en línea (comienza con un signo #, misma línea)

```
suma(a, b) # Llamamos a la función suma con los valores a y b
```

3. Comentarios en bloque (comienza con un signo #, mismo nivel de sangría)

```
# Este bloque de código calcula el área de un círculo  
# usando las variables radio y diámetro que se asignan  
# previamente en el programa.
```

## Indentación (Sangrado)

- Un grupo de sentencias que deben ejecutarse juntas
- Los distintos lenguajes representan los bloques de forma diferente

```
if (true) {  
    // ejecuta este bloque de sentencias  
} else {  
    // ejecuta otro bloque de sentencias  
}
```

- En Python los bloques están indentados **en lugar de utilizar llaves**
- Pueden estar anidados dentro de otros bloques

## Actividad 5: Corregir las sangrías en un bloque de código

- Arregla el fragmento de código en el archivo `indentacionincorrecta.py` para tener la sangría apropiada en cada bloque.
- Una vez arreglado, ejecutar el script en la terminal
- Verificar que obtienes la siguiente salida:

```
1  
2  
3  
4  
5
```

## Actividad 6: Implementar la entrada del usuario y los comentarios

- Escribe un script que tome de la entrada del usuario un número e imprima su tabla de multiplicación del 1 al 10
  1. Añadir un docstrip explicando lo que hace el script
  2. Asignar a una variable llamada número la entrada del usuario
  3. Convertir la entrada a un número entero
  4. Imprimir 25 guiones bajos como bordes superior e inferior de la tabla
  5. Imprimir la multiplicación de ese número por cada número del 1 al 10

• Resultado de la ejecución del script

Tabla de multiplicar del número: 6

```
6 * 1 = 6
6 * 2 = 12
6 * 10 = 60
```

# Guía de estilo de Python

- Favorecer la **simplicidad** en lugar de la complejidad
  - mantener y entender fácilmente
- Leer "El Zen de Python" de Tim Peters
  - 19 principios rectores que dictan el estilo de Python

```
>>> import this
```

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex  
....
```

- Entender y seguir esos principios ayudan a otros progs (entender y mantener)

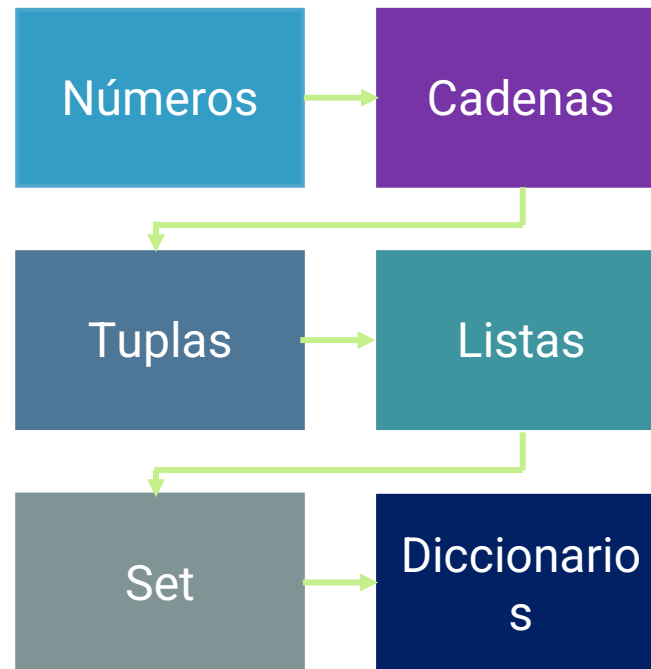


# Resumen

- Existen 2 formas de ejecutar instrucciones en Python
  - Directamente en el Shell interactivo
  - Ejecutando archivos con código previamente escrito (scripts)
- Hemos cubierto
  - Cómo estructurar instrucciones
  - Declarar y almacenar datos
  - Tipos de valores
  - Palabras reservadas
  - Función input
  - Comentarios en el código
  - Organización del código

# Tipos de datos

- Clasifican e indican al intérprete cómo el programa pretende utilizar los datos
  - las diferentes operaciones que se pueden realizar



# Datos numéricos

- Tipos de números
  - Enteros
  - Decimales (coma flotante)
- Números enteros
  - Pueden ser negativos o positivos o cero (nunca una fracción)

```
>>> entero = 49  
>>> entero_negativo = -35
```

- Precisión ilimitada
  - No hay límites en cuanto a su tamaño
  - Salva la memoria disponible

```
>>> entero_largo = 1234567898327463893216847532149022563647754227885  
439016662145553364327889985421.....
```

## Datos numéricos

- Números en coma flotante (float)

```
>>> n = 3.3333
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.7182818459045
```

- Convertir un número entero en un valor de coma flotante

```
>>> float(23)
```

# Sistemas numéricos alternativos

- Números binarios, hexadecimales y octales
- Expresados en el sistema de base (2, 16, 8)
- Para representar números utiliza:
  - [B] sólo 0s y 1s
  - [H] símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e y f
  - [O] dígitos del 0 al 7
- En Python escribe el número y se pone el prefijo
  - 0b
  - 0x
  - 0o
- Al escribir un número de este tipo en el intérprete se obtiene su equivalente decimal
- Para convertir cualquier número decimal (base 10) a estos números
  - Funciones **bin, hex y oct**

>>> 0b11 7	>>> 0xf 15
>>> 0b10 2	>>> 0x9ac 2476
	>>> 0o20 16
	>>> 0o200 128

## E4: Conversión entre diferentes tipos de sistemas numéricos

- Crear un script que tome la entrada del usuario y la convierta en un número binario, octal y hexadecimal
1. Definir la variable número que toma la entrada del usuario
  2. Convertir la entrada a un número entero
  3. Convertir el numero entero en un número binario, octal, hexadecimal
  4. Imprime los valores

```
Número para convertir: 5  
bin: 0b101  
oct: 0o5  
hex: 0x5
```

# Operadores

- Para realizar cálculos matemáticos y para asignar valores a variables
  - Aritméticos
  - De asignación
- Operadores aritméticos
  - Son funciones matemáticas que toman y realizan cálculos sobre valores numéricos
  - **Todos** los tipos numéricos de Python soportan:

- Suma: `>>> 5 + 8 + 7`
- Resta: `>>> 20 - 5`
- Multiplicación: `>>> 4 * 3`
- División: `>>> 12 / 3`
- División de enteros: `>>> 13 // 2`
- Módulo: `>>> 5 % 2`
- Exponenciación: `>>> 5 ** 3`

(da lugar a un número de punto flotante)  
(= 6, da lugar a un número de punto flotante)  
(el resto)  
(Eleva un número a una potencia)

# Operadores Aritméticos

Operador	Descripción	Ejemplo	Resultado
+	Suma	$5 + 3$	8
-	Resta	$5 - 3$	2
*	Multiplicación	$5 * 3$	15
/	División	$5 / 3$	1.6667
//	División entera	$5 // 3$	1
%	Módulo (resto de división)	$5 \% 3$	2
**	Potenciación	$5 ** 3$	125



# Operadores de asignación

- El operador de asignación simple =
- Python tiene otros operadores de asignación
  - variaciones abreviadas de los operadores simples
  - no sólo realizan una operación aritmética sino que también reasignan la variable

```
>>> x = 10
>>> x += 1
>>> print(x)
11
```

# Operadores de asignación

- Realiza la operación a la variable del lado izquierdo con el valor del lado derecho

Operador	Descripción	Ejemplo	Resultado
=	Asignación básica	x = 5	x = 5
+=	Suma y asignación	x += 3	x = x + 3
-=	Resta y asignación	x -= 2	x = x - 2
*=	Multiplicación y asignación	x *= 4	x = x * 4
/=	División y asignación	x /= 2	x = x / 2
//=	División entera y asignación	x //= 3	x = x // 3
%=	Módulo y asignación	x %= 2	x = x % 2
**=	Potenciación y asignación	x **= 2	x = x ** 2

# Orden de operaciones

- El conjunto de reglas sobre qué procedimientos deben ser evaluados primero al evaluar una expresión
- El orden en el que se evalúan los operadores es **PEMDAS**  
matemáticamente se evalúan:
  1. P (Paréntesis) Las expresiones dentro de
  2. E (Exponentes)
  3. MD (Multiplicación y División) incluyendo división entera y módulo
  4. AS (Suma y Resta)
- Las sentencias se evalúan de izquierda a derecha

## Actividad 6: Orden de las operaciones

- Reescribe la siguiente ecuación como una expresión de Python y obtén el resultado

$$5(4 - 2) + \left( \frac{100}{\frac{5}{2}} \right) 2$$

## Actividad 8: Uso de operadores aritméticos

- Escribir un script que tome como entrada del usuario el número de días y lo convierta en años, semanas y días (ignorar los años bisiestos)
1. Declarar la entrada del usuario
  2. Convertir a un número entero
  3. Calcular el número de años en ese conjunto de días
  4. Los días restantes que no fueron convertidos a años convertirlo en semanas
  5. Los días restantes que no fueron convertidos a semanas dejarlo en días
  6. Imprimir el resultado

Número de días: 600  
años: 1  
semanas: 33  
días: 4

## Cadenas de caracteres (*strings*)

- Son una secuencia de caracteres
- Pueden ir entre comillas simples (') o dobles (")

```
>>> "una cadena"  
'una cadena'  
>>> 'otra cadena'  
'otra cadena'
```

- Una cadena con comillas dobles puede contener comillas simples y viceversa

```
>>> "Car's"  
"Car's"
```

```
>>> "¡Socorro!", exclamó  
"¡Socorro!", exclamó.'
```

## Cadenas de caracteres (*strings*)

- Entre comillas triples se encierra los caracteres de una cadena multilínea

```
>>> s = """Esta es una cadena multilínea
... 1
... 2
... 3
... fin de la cadena multilínea"""
>>> s
'Esta es una cadena multilínea\n1\n2\n3\nfin de la cadena multilínea'
```

# Operaciones con cadenas de caracteres

1. Repetir cadenas (operador **\***)

```
>>> print("-" * 25)
```

2. Concatenar cadenas (operador **+**)

```
>>> "I "+"love "+"Python"  
'I love Python'
```

- Las cadenas son inmutables

```
>>> hola = 'Hola'  
>>> mundo = 'Mundo'  
>>> hola = hola + mundo  
HolaMundo
```



# Operaciones con cadenas de caracteres

## 3. Indexación

- de izquierda a derecha
- de derecha a izquierda

0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
I		l	o	v	e		P	y	t	h	o	n

```
>>> s="I love Python"
>>> s[0]

>>> s[10]

>>> s[-10]
```

\* Un índice con un carácter que no existe Python lanzá un IndexError.

# Operaciones con cadenas de caracteres

- 4. Obtener una **rebanada/subcadena** dentro de un rango de índices

```
>>> cadena = "Bioelectromagnetismo"  
>>> cadena[0:3]
```

`cadena[índice_inicial : índice_final]`

```
>>> cadena[:3]
```

```
>>> cadena[10:20]
```

```
>>> cadena[10:]
```

```
>>> cadena[3:10]
```

```
>>> cadena[-17:-10]
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1
0	9	8	7	6	5	4	3	2	1	0									
B	i	o	e	l	e	c	t	r	o	m	a	g	n	e	t	i	s	m	o

## Actividad 9: Rebanar cadenas (subcadenas)

- Dadas las siguientes sentencias ¿cuál será la salida?

```
1. >>> "[8,9,10]" [3]
2. >>> "La curiosidad por explorar la vida" [7]
3. >>> "La curiosidad por explorar la vida" [12:19]
4. >>> "más allá de las estructuras" [9:]
5. >>> "más allá de las estructuras"[:-1]
```

# Operaciones con cadenas de caracteres

## 5. Longitud = número de caracteres que contiene

- utilizando la función `len()`
- toma una cadena como parámetro y devuelve un número entero

```
>>> pregunta = "¿Quién fue el primero de los Beatles que dejó el grupo?"  
>>> len(pregunta)  
55
```

## 6. Formato: construir nuevas cadenas utilizando valores existentes

- 6.1 Interpolación de cadenas
- 6.2 Método `str.format()`
- 6.3 Formato `%`.

# Operaciones con cadenas de caracteres

## 6.1 Interpolación de cadenas

- Utiliza cadenas formateadas
  - Van precedidas de una **f** para indicar cómo deben interpretarse
- Para insertar una variable, **se colocan llaves** que contienen la expresión que queremos poner dentro de la cadena

```
>>> postre='helado'
>>> f"Comí un poco de {postre} y estaba buenísimo"
'Comí un poco de helado y estaba buenísimo'
```

```
>>> numero=7
>>> f"{numero*2} es sólo un número"
'14 es sólo un numero'
```

# Operaciones con cadenas de caracteres

## 6.2 El método `str.format()`

- Ponemos llaves en las posiciones donde queremos poner nuestros valores
- Llamamos al método `format`
  - toma el argumento (nuestra variable)
  - sustituye las llaves por el valor

```
>>> reduccion=43
>>> año=2030
>>> cadena="En el año {} las emisiones se reducirán un {}%".format(año,reduccion)
>>> cadena
'En el año 2030 las emisiones se reducirán un 43%'
```

# Métodos con cadenas de caracteres

Método:	Devuelve una copia de la cadena con:
str.capitalize()	la primera letra en mayúscula y el resto en minúscula
str.lower()	todos los caracteres convertidos a minúsculas
str.upper()	todos los caracteres convertidos a mayúsculas:
str.startswith()	comprueba si una cadena empieza por el prefijo especificado
str.endswith()	comprueba que la cadena termina con el sufijo especificado
str.strip()	Elimina los caracteres iniciales y finales sin argumento -> elimina todos los espacios en blanco
str.replace()	sustituye todas las apariciones por la nueva

# Métodos con cadenas de caracteres

```
>>> "---mi-editor-favorito-es-VS-Code-----".strip("-")
'mi-editor-favorito-es-VS-Code'

>>> ---mi-editor-favorito-es-VS-Code-----".replace("-", " ")
' mi editor favorito es VS Code '

>>> "---mi-editor-favorito-es-VS-Code-----".replace("VS-Code","Notepad++")
'---mi-editor-favorito-es-Notepad++-----'
```



## Actividad 10: Trabajar con cadenas

Escriba un script que **convierta las últimas n letras de una cadena a MAYÚSCULAS** debe tomar como entrada del usuario

- la cadena a convertir
- un número entero (las últimas n letras a convertir)

1. Solicitar al usuario la cadena a convertir
2. Solicitar el número de últimas letras a convertir
3. Dividir la cadena, y obtener la primera parte
4. Obtener la última parte de la cadena (la que vamos a convertir)
5. Transformar la subcadena
6. Concatenar la primera y la nueva parte
7. Ejecuta el script

Cadena para convertir: tipos de datos compuestos  
¿Cuántas últimas letras hay que convertir? 5  
tipos de datos compuESTOS

## Actividad 11: Manipulación de Cadenas

Escriba un script que cuente el número de ocurrencias de una palabra especificada en una frase dada

1. Tomar de la entrada del usuario la frase y la palabra a buscar
2. Formatear la entrada eliminando los espacios en blanco y convirtiéndola en minúsculas
3. Cuente las ocurrencias de la subcadena
4. Imprime los resultados

Frase: En medio del camino de nuestra vida me encontré en un oscuro bosque  
Palabra a buscar en la frase: en  
Hay 3 ocurrencias de 'en' en la frase.

## Listas –arrays–

- Son un tipo de datos agregados
- Son heterogéneas
- Son similares a las cadenas
  - los valores dentro de ellas están indexados
  - tienen una propiedad de longitud y conteo
- Son mutables
- Las listas se hacen con **elementos separados por comas y encerrados entre corchetes**

```
>>> digitos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> letras = ["a", "b", "c", "d"]
>>> lista_mixta = [1, 3.14159, "Primavera", "Verano", [1, 2, 3, 4]]
```

# Operaciones con listas

Operación	Función	Ejemplo
Tamaño	Obtiene el número de elementos de una lista	<code>len(lista)</code>
Trocear	Devuelve una nueva lista derivada de la anterior	<code>lista[0:2]</code>
Concatenar	Puedes sumar dos listas utilizando el operador +	<code>lista1 + lista2</code>
Cambiar valores	Asignar el nuevo valor a lo que esté en ese índice	<code>lista[1] = 25</code>
Insertar	Insertar un valor al final de una lista	<code>lista.append(4)</code>
Ordenar	Ordenar la lista	<code>lista.sort()</code>
Invertir	Invertir el orden sin ordenar	<code>lista.reverse()</code>

## E5: Referencias a listas

- Cree una nueva lista `>>> lista1 = [1, 2, 3]`
- Luego, asigna una nueva variable, lista2 a lista1: `>>> lista2 = lista1`
- Añade 4 a la lista2 y comprueba el contenido de la lista1  

```
>>> lista2.append(4)
>>> lista1
[1, 2, 3, 4]
```
- Inserta el valor 'a' en el índice 0 de la lista1 `>>> lista1[0] = "a"`
- Comprueba el contenido de la lista2  

```
>>> lista2
['a', 2, 3, 4]
```

## Actividad 12: Trabajar con listas

Escribir un script que **obtenga los n primeros elementos de una lista**

1. Crea la lista con los siguientes elementos: 57, 16, 37, 83, 571, 163, 9, 41, 27
2. Imprimir la lista
3. Leer la entrada del usuario con el número de elementos a obtener
5. Imprimir la porción de la lista desde el primer hasta el n-elemento
6. Ejecute el script

```
Lista: [57, 16, 37, 83, 571, 163, 9, 41, 27]  
Número de elementos a recuperar de la lista: 3  
[57, 16, 37]
```

## E6: Ejercicios con listas

Ejercicio	Operación	Ejemplo
1	Crear una lista vacía	<code>países = []</code>
2	Insertar elementos	<code>países = ["España", "China", "Rusia"]</code>
3	Llamar elemento	<code>países[0]</code>
4	Lista de listas	<code>[["Valencia", "Sevilla"], "España", "China", "Rusia"]</code>
5	Rebanar ( <i>Slicing</i> )	<code>países[0][2]</code>
6	Obtener la longitud	<code>len(países)</code>
7	Cambiar los valores de los elementos	<code>países[2]="Alemania"</code>

## E6: Ejercicios con listas

Ejercicio	Operación	Ejemplo
8	Eliminar	<b>del</b> paises[2]
9	Pertenencia	"China" <b>in</b> paises
10	Asignación de un elemento a una variable	result=paises[1]
11	Encontrar valores index()	paises. <b>index</b> ("Japón")
12	Añadir valores a una lista con append()	paises. <b>append</b> ("Canada")
13	Añadir valores a una lista con insert()	paises. <b>insert</b> (3,"Mexico")
14	Eliminar un elemento	paises. <b>remove</b> ("Japón")
15	Ordenar elementos	paises. <b>sort</b> ()



# Booleanos

- Solo puede tener dos valores
  - Verdadero o Falso

```
>>> True
True
>>> type(True)
<class 'bool'>
```

- Se asocian con las sentencias de control
  - Cambian el flujo del programa
  - Dependiendo de la veracidad de las cantidades

# Operadores de comparación

- Comparan valores de los objetos, objetos, e identidades
- Los objetos no necesitan ser del mismo tipo
- Hay ocho operadores de comparación en Python:

Operador	Descripción	Ejemplo	Resultado
==	Igual a	5 == 5	True
!=	Diferente de	5 != 3	True
>	Mayor que	5 > 3	True
<	Menor que	5 < 3	False
>=	Mayor o igual que	5 >= 5	True
<=	Menor o igual que	5 <= 3	False
is	Identidad de objeto	True is True	True
is not	Identidad negada	True is True	False

# Operadores Lógicos

- Combinan expresiones booleanas

Operador	Descripción	Ejemplo	Resultado
and	Devuelve True si <b>ambas</b> expresiones son True	(5 > 3) <b>and</b> (8 > 6)	True
or	Devuelve True si <b>al menos una</b> expresión es True	(5 > 3) <b>or</b> (8 < 6)	True
not	Invierte el valor lógico de una expresión	<b>not</b> (5 > 3)	False

- **and** Evalúa el segundo argumento SI el primero es T
- **or** Evalúa el segundo argumento SI el primero es F

# Operadores de pertenencia

- Los operadores **in** y **not in** comprueban la pertenencia
- Todas las secuencias (listas y cadenas) admiten este operador
- Recorren cada elemento para ver si el elemento que se busca está dentro de la lista
- Los valores de retorno son **True** o **False**

```
>>> numeros = [1,2,3,4,5]
>>> 3 in numeros
True
>>> 6 in numeros
False
```

# Tuplas

- Son listas inmutables
- Se denotan utilizando paréntesis en lugar de los corchetes
- Creación
- Operaciones

```
>>> deportes1 = ('fútbol', 'natación', 'ciclismo')  
>>> deportes2 = 'fútbol', 'natación', 'ciclismo'
```

```
>>> deportes1 + deportes2  
>>> deportes3=(deportes1, deportes2)  
>>> deportes1=deportes1*3  
>>> deportes1[2]  
>>> deportes1[2]='running'  
>>> deportes1=deportes1[2:4]  
>>> del deportes1
```

# Diccionarios

- Permiten acceder a cualquier valor utilizando la clave
- Se indexan mediante claves (cadenas)
- Hay dos tipos de diccionarios
  - **Dict**: diccionario estándar, no ordenado

```
>>> d = {"a": 1, "b": 2, "c": 3}
```

- **OrderedDict**: mantiene el orden de inserción
  - útil cuando el orden es fundamental

```
>>> od = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```



# Operaciones con diccionarios

Operación	Descripción	Código
<b>Crear un diccionario vacío</b>	Define un diccionario vacío	<code>diccionario = {}</code>
<b>Crear un diccionario con valores</b>	Define un diccionario con pares clave-valor	<code>diccionario = {"nombre": "Juan", "edad": 30}</code>
<b>Acceder a un valor</b>	Devuelve el valor asociado a la clave "nombre"	<code>diccionario["nombre"]</code>
<b>Acceder de forma segura</b>	Evita errores si la clave no existe	<code>diccionario.get("profesión", "No especificado")</code>
<b>Añadir o actualizar un valor</b>	Agrega o modifica una clave-valor	<code>diccionario["ciudad"] = "Madrid"</code>
<b>Eliminar un valor</b>	Elimina el par clave-valor asociado a "ciudad"	<code>del diccionario["ciudad"]</code>
<b>Eliminar y obtener un valor</b>	Elimina la clave "edad" y devuelve su valor	<code>valor = diccionario.pop("edad")</code>

# Operaciones con diccionarios

Operación	Descripción	Código
<b>Obtener la longitud</b>	Devuelve el número de pares clave-valor	<code>len(diccionario)</code>
<b>Comprobar existencia de clave</b>	Devuelve True si la clave existe, False en caso contrario	<code>"nombre" in diccionario</code>
<b>Copiar un diccionario</b>	Crea una copia superficial del diccionario.	<code>copia = diccionario.copy()</code>
<b>Obtener solo claves</b>	Devuelve un objeto con las claves del diccionario.	<code>diccionario.keys()</code>
<b>Obtener solo valores</b>	Devuelve un objeto con los valores del diccionario.	<code>diccionario.values()</code>
<b>Obtener pares clave-valor</b>	Devuelve pares clave-valor como tuplas.	<code>diccionario.items()</code>

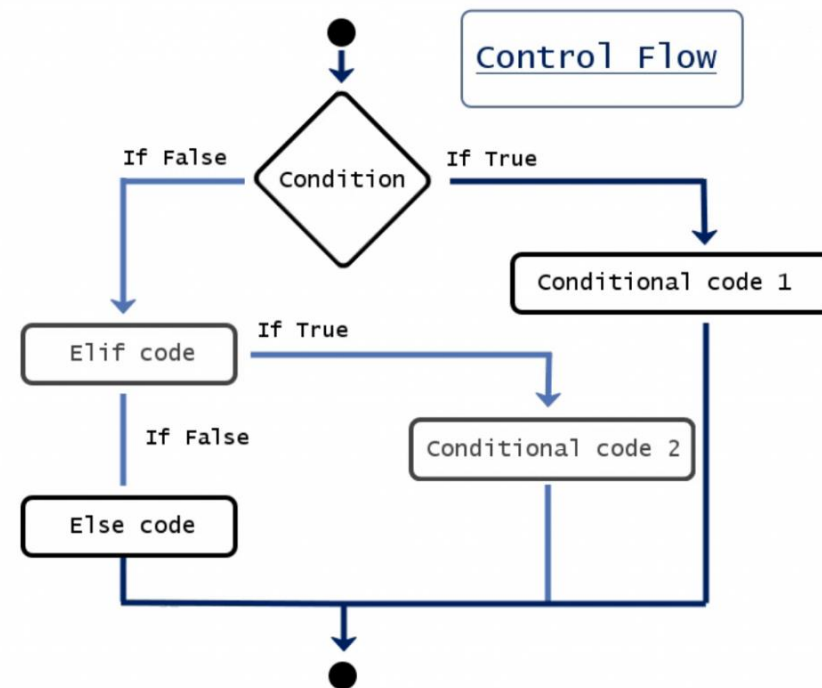


# Resumen Tipos de datos

- Conceptos clave en Python
  - Tipos de **datos básicos**: Números, cadenas, listas, booleanos y diccionarios
  - Operadores **numéricos**: Operaciones y manipulaciones con datos numéricos
  - **Cadenas**: Indexación, rebanado y formateo de texto
  - **Listas** (arrays): Creación, manipulación y acceso a elementos
  - **Booleanos** y operadores **lógicos**: Comparaciones y condiciones
  - **Diccionarios**: Tipos como ``dict`` y ``OrderedDict``, y atributos asociados

# Sentencias de control de flujo

- Manejo de sentencias de control
- Control del flujo de ejecución
- Ejecución repetitiva



# Objetivos de aprendizaje

Ser capaz de:

- Identificar las sentencias de control y sus usos
- Modificar el flujo de ejecución de un programa
- Utilizar estructuras de bucle para automatizar tareas
- Implementar bifurcaciones dentro de bucle para mayor flexibilidad



# Flujo del programa

Describe cómo se ejecutan las sentencias en el código  
Considerando la prioridad de los diferentes elementos

- **Secuencial:** De arriba hacia abajo
- **Orden de ejecución:** Cada línea espera a que las líneas anteriores completen su ejecución antes de continuar

# Sentencia de control

Es una **estructura en el código** que **cambia condicionalmente el flujo** del programa

- Permite ejecutar diferentes partes del código
- Permite ejecutar un bloque de **código repetidamente**

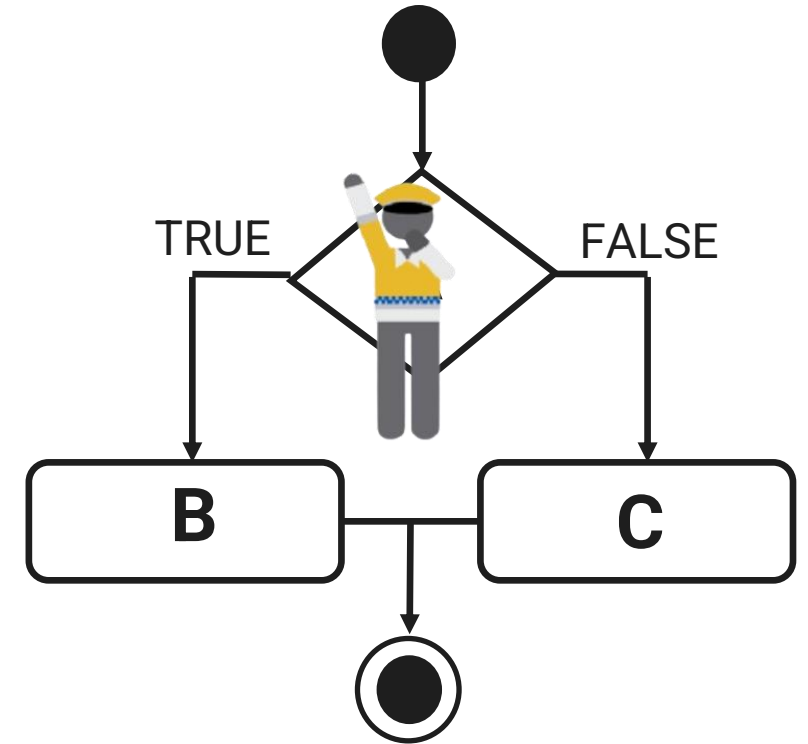
Principales sentencias de control:

- IF: Para tomar decisiones
- WHILE: Para repetir bloques de código

# La sentencia IF

- Si una condición es verdadera
  - Ejecutar un bloque de código
  - caso contrario ejecuta un bloque alternativo
- La cláusula **else** es opcional
- Se puede **encadenar** varias sentencias IF
- Sintaxis básica:

```
if condición:  
    #Ejecuta si es True (T)  
else:  
    #Ejecuta si es False (F)
```



## E7: Uso de la sentencia **if**

1. Declarar una variable llamada **año\_de\_lanzamiento** y asignarle el valor **"1991"**
2. Declarar otra variable llamada **respuesta** y asígnale la respuesta del usuario a la pregunta: *"¿Cuándo se lanzó Python por primera vez?"*
3. Utilizar **if** para comprobar si la respuesta ingresada es correcta
4. Utilizar **elif** para verificar si la respuesta es mayor o menor que la correcta e informar al usuario si su respuesta es *"demasiado alta / baja"*
5. Imprimir un mensaje de salida como *"¡Adiós!"*

## Actividad 13: Trabajando con la sentencia **if**

Modificamos el ejercicio anterior como sigue:

1. La pregunta al usuario debe ser

"Escribe TRUE o FALSE: ¿Python se lanzo en 1991?"

2. Si el usuario ingresa TRUE, imprime "Correcto"

3. Si el usuario ingresa FALSE, imprime "Incorrecto"

4. Cualquier otra entrada del usuario debe imprimir el mensaje

'Por favor responda TRUE o FALSE '

```
Escribe TRUE o FALSE: ¿Python se lanzo en 1991?: SI
Por favor responda TRUE o FALSE
Adios!
```

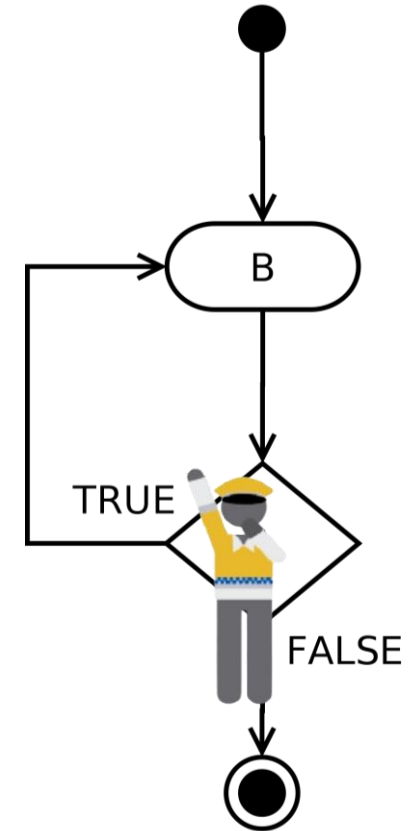
```
Escribe TRUE o FALSE: ¿Python se lanzo en 1991?: FALSE
Incorrecto
Adios!
```



# La sentencia WHILE

- Permite ejecutar un bloque de código repetidamente mientras una condición permanezca verdadera
- Sintaxis básica:

**while** condición:  
    # mientras la condición sea verdadera  
else:  
    # una vez que la condición ya no es verdadera  
    # se ejecuta este código una sola vez



## E8: Uso de la sentencia while

1. Declarar una variable llamada **contador** con valor “1”
2. Declarar la variable **final** con un valor de 10
3. Escribir un **while** con la condición de que el contador sea menor o igual que el valor de la variable final
4. Para cada valor del contador, imprimirlo e incrementarlo en 1
5. Cuando la condición ya no es verdadera, imprimir “*Has llegado al final*”

```
1
2
3
4
5
6
7
8
9
10
Ha llegado al final
```

## E9: Uso de while para mantener un programa en ejecución

Reescribir el programa que escribimos antes y añadiremos una sentencia while:

1. Declarar `año_de_lanzamiento` en 1991 (respuesta correcta)
2. Establecer una variable “correcto” con el valor False.
3. Mientras la respuesta proporcionada no sea correcta mantener el programa en marcha
  - Imprimir la pregunta en la terminal “En que año...”
  - Esperar la respuesta del usuario
5. Usar una sentencia `if` para comprobar que la respuesta es correcta
  - Si la respuesta es correcta
    - Imprimir un mensaje de éxito
    - Cambiar el valor de correcto a True (para salir del bucle)
  - Si la respuesta es incorrecta
    - Animar al usuario a volver a intentarlo
6. Al final del programa imprimir un mensaje de salida

# Actividad 14: Trabajar con la sentencia while

Crear un **programa de autenticación de contraseñas** utilizando un bucle while

1. Definir una variable contraseña con el valor “**random**”
2. Definir un **validador** booleano con el valor **False**
3. Inicie un bucle while que continúe solicitando la entrada del usuario mientras el validador sea **False**
4. Valide la contraseña ingresada
  - Si la contraseña es **correcta**
    - Imprimir un mensaje de bienvenida al usuario
    - Establecer el valor del validador a True
  - Si la respuesta es **incorrecta**
    - Animar al usuario a volver a intentarlo
5. Ejecutar el script

```
por favor introduzca su contraseña: random
Contraseña incorrecta, intentelo nuevamente...
por favor introduzca su contraseña: random
Bienvenido usuario!
```

## while versus if

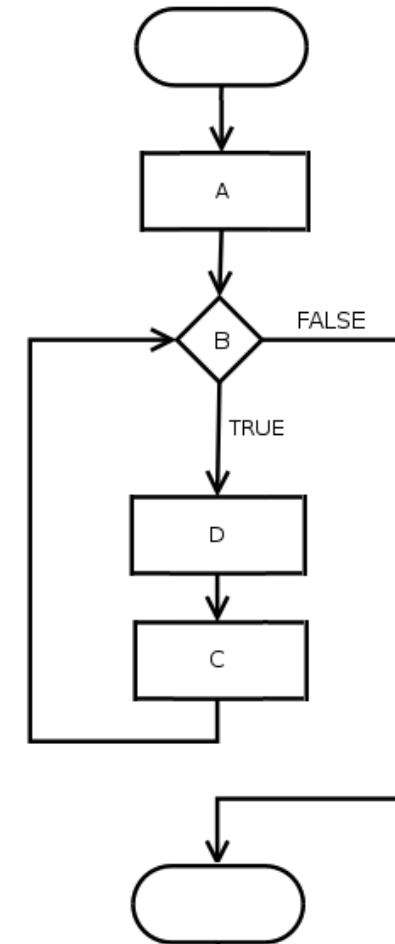
- **if** da la oportunidad de bifurcar la ejecución del código
- El código dentro del bloque if se ejecuta una sola vez
- **while** permite ejecutar un bloque de código varias veces

# Bucles

- Permiten ejecutar un bloque de código varias veces de forma eficiente
- Se utilizan para iterar sobre cualquier objeto
  - Cadenas, Listas, Diccionarios, Archivos
- Un iterable es una colección de elementos homogéneos agrupados
  - Los elementos individuales comparten las mismas propiedades
  - Al combinarse, forman una entidad más compleja

# El bucle FOR

- Permite recorrer y ejecutar un bloque de código para cada elemento de un iterable
- **for** VS **while**:
  - **for** el bloque de código se ejecuta un número predeterminado de veces, iterando sobre cada elemento del iterable
  - **while** el código se ejecuta un número arbitrario de veces siempre que se cumpla una condición específica



## E4: Uso del bucle for

1. Declarar una variable llamada **números** e inicializarla con una lista de enteros del 1 al 10
2. Recorrer la lista utilizando un bucle for  
Crear una variable llamada **“num”**
3. Dentro del bucle  
Calcula el cuadrado del numero  
Asignamos el resultado a una variable **“cuadrado”**
4. Imprimimos un mensaje que muestre el valor  
*“El cuadrado de **num** es: **cuadrado**”*

```
El cuadrado de 1 es: 1
El cuadrado de 2 es: 4
El cuadrado de 3 es: 9
El cuadrado de 4 es: 16
El cuadrado de 5 es: 25
El cuadrado de 6 es: 36
El cuadrado de 7 es: 49
El cuadrado de 8 es: 64
El cuadrado de 9 es: 81
El cuadrado de 10 es: 100
```



# La función Range

- Genera una secuencia de números útil para iterar con un bucle for
- Sintaxis:  
    `range([inicio], stop, [paso])`  
    inicio: Número inicial de la secuencia (0 si no se especifica)  
    stop: Generar números hasta este valor, pero **sin incluirlo**  
    paso: La diferencia entre cada número consecutivo de la secuencia
- Para ver los números convierte el objeto **range** a un objeto **lista**

## Actividad 15: El bucle for y la función range

- Usando un bucle **for** y una función de **rango**, encuentra los números pares entre 2 y 100 y luego calcula su suma

- La salida será:

2550

1. Definir una variable para almacenar la suma total
2. Crear un bucle **for** con un **rango** para los números pares entre 2 y 100.
3. Sumar cada número del bucle a la variable suma
4. Imprime la suma total cuando el bucle ha finalizado



## Anidación de bucles

- Es la práctica de colocar un bucle dentro de otros bucles
  - Cuando se necesita acceder a datos dentro de una estructura de datos compleja
  - Para alcanzar el nivel de granularidad requerido

## E11: Anidamiento de bucles

- Utilizaremos bucles anidados para los cuadrados de los números:
  1. Crear una variable llamada “grupos”, que es una lista que contiene tres sublistas, cada una con tres enteros  
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  2. Iterara sobre la lista grupos utilizando un bucle
  3. Por cada sublista, realizar otro bucle para acceder a cada número, al que llamaremos “num”
  4. Calcular el cuadrado de cada número
  5. Imprimir el cuadrado del número

```
1 su cuadrado es: 1
2 su cuadrado es: 4
3 su cuadrado es: 9
4 su cuadrado es: 16
5 su cuadrado es: 25
6 su cuadrado es: 36
7 su cuadrado es: 49
8 su cuadrado es: 64
9 su cuadrado es: 81
```

## Actividad 16: Bucles anidados

- Suma los números pares e impares entre 1 y 11 e imprima el cálculo
1. Escribir un **for** con la función de rango para los números pares
  2. Escribir otro **for** con la función de rango para los impares
  3. Utilizar una variable **suma** para calcular la suma de pares e impares
  4. Imprimir **suma**

```
2 + 1 = 3
2 + 3 = 5
2 + 5 = 7
2 + 7 = 9
2 + 9 = 11
4 + 1 = 5
4 + 3 = 7
....
8 + 7 = 15
8 + 9 = 17
10 + 1 = 11
10 + 3 = 13
10 + 5 = 15
10 + 7 = 17
10 + 9 = 19
```

# Resumen Sentencias de control de flujo

- Lo que hemos aprendido
  - El flujo de un programas
  - Control y bifurcación del flujo de un programa (if y while).
  - Aplicaciones prácticas de las dos sentencias de control
  - Diferencias en implementación y sintaxis
  - Estructuras de bucles
  - La función range
  - Anidación de bucles

# Funciones

- Agrupan líneas de código que implementan alguna funcionalidad específica
- Reutilización: Cuando un fragmento de código se repite en varias partes del programa
- Abstracción de código complejo, dividiendo el programa en mini-programas dentro de un programa más grande
- Es recomendable escribir funciones que realicen una tarea específica en lugar de incluir demasiada funcionalidad en una sola función
- Facilitan la modularización del código mejorando la mantenibilidad y la facilidad de depuración

# Objetivos de aprendizaje

Ser capaz de:

- Describir los distintos tipos de funciones en Python
- Definir las variables globales y locales
- Definir una función que tome un número variable de argumentos



# Funciones incorporadas en Python

- El intérprete de Python tiene una serie de funciones incorporadas que están siempre disponibles
- Se pueden utilizar en cualquier parte del código sin necesidad de ninguna importación
- Ejemplos:
  - `input([prompt])`: Lee una línea de la entrada
  - `print()`: Imprime objetos
  - `map()`: Devuelve un iterador

# Funciones definidas por el usuario

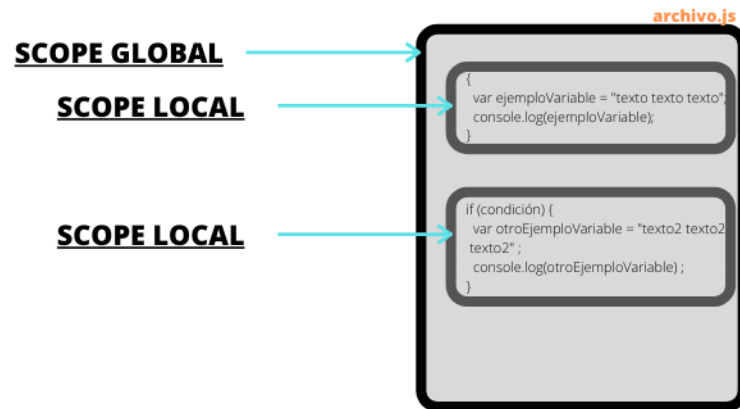
- Ayudan a lograr un objetivo específico
- El uso principal es organizar los programas en fragmentos lógicos
- Sintaxis:

```
def nombre_de_la_función( parámetro1, parámetro2, parámetroN ):
    # La lógica va aquí
    return
```

- A menudo se llaman desde otras funciones
- Los nombres de los argumentos que pasamos no tienen que coincidir con los nombres de los parámetros esperados por la función

# Variables globales y locales

- Variables locales
  - Se definen dentro del cuerpo de una función
  - Sólo son accesibles dentro de la función (ámbito local)
- Variables globales
  - Se definen fuera del cuerpo de una función
  - Son accesibles tanto fuera como dentro de las funciones (ámbito global)



## E12: Definición de variables globales y locales

1. Define una variable global, **numero** inicializada a 5
2. Define una función llamada **suma** que toma dos parámetros de nombre: primero y segundo
3. Dentro de la función, suma los dos parámetros que se han pasado y la variable global **numero**, devuelve el total
5. Llama a la función de suma con dos parámetros (10 y 20)
6. Imprime el valor del número y el de la variable total

El primer número que inicializamos fue 5  
El total después de la suma es 35

## Uso de main()

- Indica al sistema operativo qué código debe ejecutar cuando se invoca al programa
  - Requerido la mayoría de lenguajes de programación
  - No es necesario en Python
- Ayuda a estructurar un programa de forma lógica
- El programa se ejecutará por sí mismo, de forma autónoma
- Se define algunas variables especiales
  - `__name__` y se establecerá automáticamente a `__main__`

```
if __name__ == "__main__":  
    main()
```

## E13: Restructurar el código con main()

1. Restructurar el ejercicio anterior con la función main()

Ejemplo de la estructura:

```
def suma():  
  
def main():  
  
if __name__ == "__main__":  
    main()
```

# Parámetros de la función

- Son la información que se necesita pasar a la función
- Los argumentos son a las funciones como los ingredientes son a una receta
- Tipos de argumentos:
  - Requeridos (tienen que estar presentes cuando se llama a una función)
  - Palabra clave (identificando los argumentos por sus nombres)
  - Por defecto (asignar un valor por defecto )
  - Número variable de argumentos  
(reciba cualquier número de variables, *\*args*)

# Tipos de argumentos en parámetros de la F

## # Argumentos requeridos

```
def divi(primerο, segundo):  
    return primerο/segundo  
  
cociente = divi(10, 2)
```

## # Argumentos de palabras clave

```
def divi(primerο, segundo):  
    return primerο/segundo  
  
cociente = divi(segundo=2, primerο=10)
```

- # Argumentos por defecto

```
def divi(primerο, segundo=2):  
    return primerο/segundo  
  
cociente = divi(10, 5)
```

## # Número variable de argumentos

```
def suma(*args):  
    total = 0  
    for i in args:  
        total += i  
    return total  
  
respuesta = suma(20, 10, 5, 1)
```



## Actividad 17: Argumentos de la función

- Escribe una función que reciba **n argumentos**
  - Utiliza la sentencia *continue* para saltar los enteros e imprime todos los demás valores.
1. Definir una función llamada **imprimir\_argumentos** recibe un número variable de argumentos
  2. Utilice un bucle for para iterar sobre los argumentos
  3. Compruebe si el valor del elemento es de tipo entero  
Si lo es, utilice la sentencia *continue* para ignorarlo.
  4. Imprime los argumentos que no son enteros

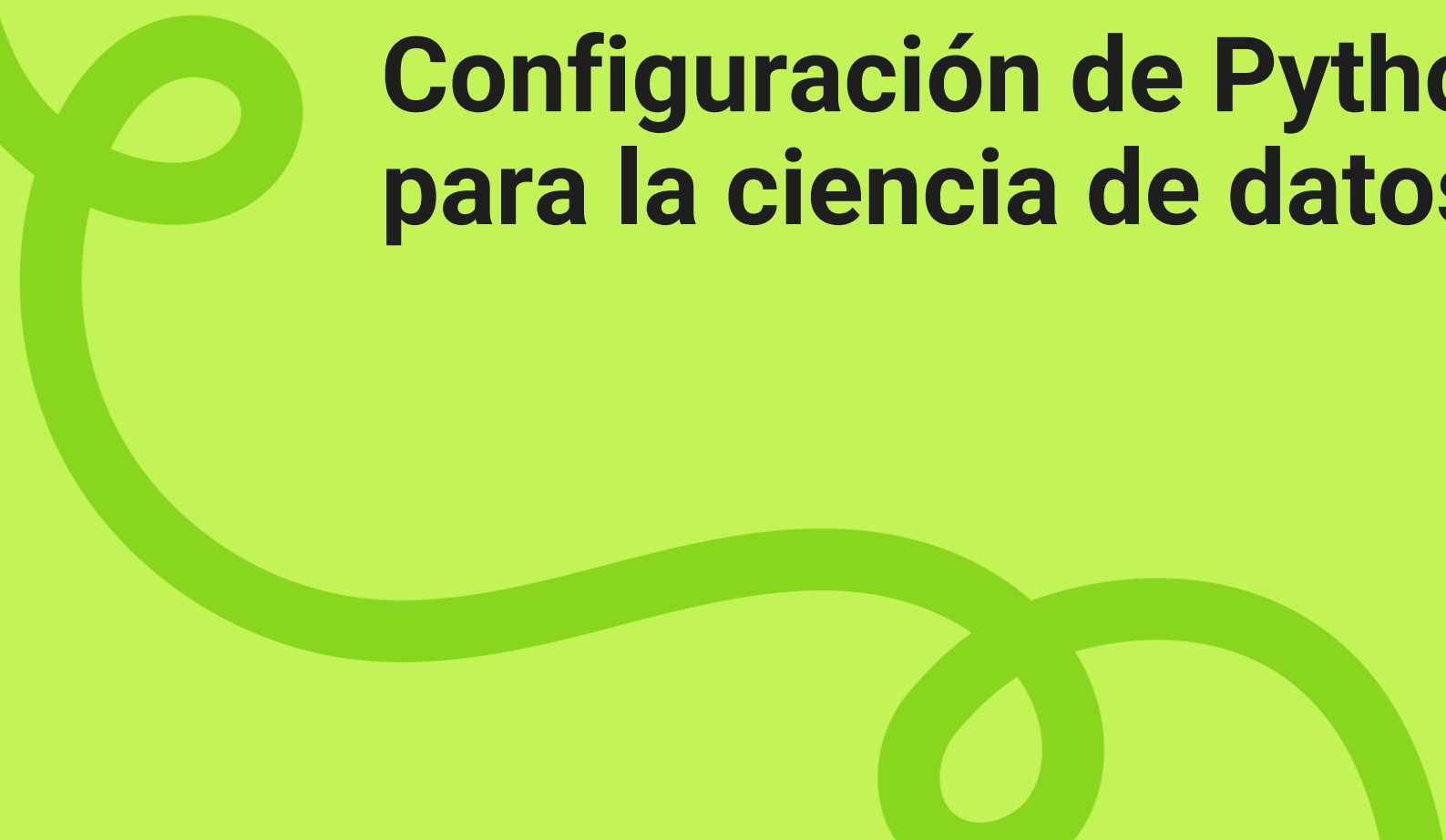
```
imprimir_argumentos("a",1,"b",2,"c",3)
```

a  
b  
c

# Resumen Funciones

Hemos aprendido

- Los distintos tipos de funciones, sus diferencias, sintaxis y casos de uso
- Cómo y dónde aplicarlas para resolver problemas específicos
- Cómo se pueden usar para ayudar a dividir programas en subprogramas
- El uso de funciones permite:
  - Reutilizar código y evitar la repetición
  - Hacer programas más organizados y fáciles de mantener

A thick, vibrant green line forms a decorative swirl on the left side of the slide. It starts from the bottom, loops upwards and to the left, then curves back down and to the right, ending near the bottom center.

## **Unidad 3: Configuración de Python para la ciencia de datos**

# Trabajando con Google Colab

- Servicio gratuito basado en la nube de Google
- Brinda acceso a estudiantes y grupos de bajos recursos en todo el mundo
- Replica la funcionalidad de Jupyter Notebook
- Es ideal para el aprendizaje automático, la ciencia de datos y la educación
- Permite trabajar desde navegadores web
- No es necesario instalar software en tu dispositivo
- Utiliza archivos .ipynb

# Ventajas de Google Colab

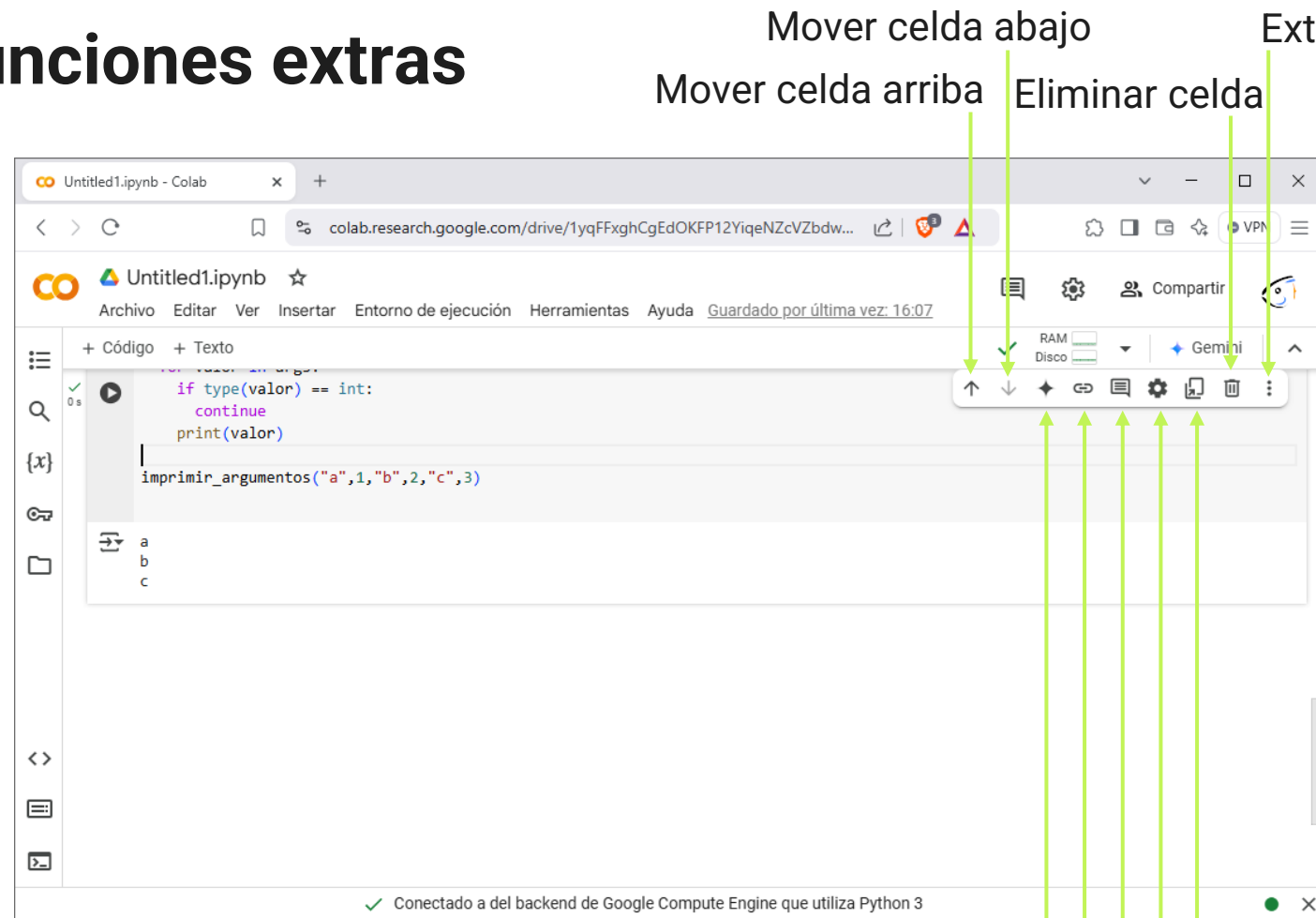
- Accesibilidad global:
  - Accede a tus archivos desde cualquier dispositivo sincronizado
  - Compatible con tablets y smartphones (aunque menos práctico)
- Colaboración en tiempo real:
  - Comparte y edita código con otros usuarios al instante
- Integración en la nube:
  - Compatible con Google Drive y GitHub
- Entorno amigable:
  - Similar a Jupyter Notebook, con diferencias menores



## Tareas Comunes con Google Colab

- Crear y ejecutar celdas de código
- Documentar con celdas de texto
- Crear gráficos y analizar datos
- Guardar archivos en Google Drive o GitHub
- Descargar archivos .ipynb o .py

# Menú de funciones extras



Copiar enlace a celda  
Funciones de IA disponibles  
Replicar celda en una pestaña  
Abrir configuraciones del editor  
Agregar un comentario

## Creando celdas de texto

- Similar a celdas Markdown en Jupyter Notebook
- Interfaz gráfica para facilitar el formato:
  - Íconos para encabezados, listas y otros elementos
- Menú lateral con opciones como en las celdas de código
- Las celdas de texto no se pueden ejecutar para resolver su marcado



## Celdas especiales

- Accesibles desde el menú Insertar
  - Ejemplo: Encabezados de sección
- Celda creada con el nivel de encabezado adecuado
- Permite un formato rápido y profesional

# Ejecutar el código

Opciones en el menú “Entorno de Ejecución”:

- Ejecutar la celda actual o varias celdas
- Ejecutar todas las celdas: Inicia desde la primera celda con código
- Interrumpir ejecución: Detiene cualquier proceso en ejecución
- Reiniciar entorno: Limpia el estado del notebook para verificar el funcionamiento del código desde cero

# Visualizando notebooks

- Icono de Tabla de contenidos en la barra lateral:
  - Permite navegar fácilmente entre secciones
  - Opción: Agregar una nueva sección (+ Sección)
- Información del notebook:
  - Tamaño del archivo
  - Configuración y propietario

## Métodos de compartir notebooks

- Enviar un mensaje con el enlace del notebook
- Obtener un enlace y compartirlo directamente
- Opciones avanzadas de permisos (edición, solo lectura, etc.)
- Integración con GitHub y Gists.

## No permite en Colab

- Hosting de archivos
- Entrega de contenido multimedia
- Cualquier servicio web no relacionado con el procesamiento interactivo
- Descargar torrents o compartir archivos entre pares
- Conexión a proxies remotos
- Minería de criptomonedas
- Ejecutar ataques de denegación del servicio
- Piratería informática de contraseñas
- Usar múltiples cuentas para evitar restricciones de acceso o de recursos
- Crear deepfakes

# Programación con IA

- Funciones avanzadas de programación con IA incluyen:
  - Autocompletado basado en IA
  - Generación de código a partir de lenguaje natural
  - Chatbot de Gemini: Un asistente interactivo para preguntas más generales sobre Python
- Proporciona explicaciones junto con fragmentos de código
- Funciona mejor con Python y está optimizada para ese lenguaje
- Es experimental y es posible que algunas de las respuestas sean imprecisas
- Mejora día tras día

# **Unidad 4:**

## **Trabajar con datos**

# Trabajar con Datos Reales

- Los datos son esenciales para las aplicaciones de ciencia de datos
  - Están dispersos, en distintos formatos y difíciles de interpretar
  - Cada organización almacena y gestiona datos de manera diferente
- Dominar técnicas para acceder y trabajar con datos en múltiples formatos y ubicaciones
  - Manipulación de flujos de datos
  - Trabajo con archivos planos y no estructurados
  - Interacción con bases de datos relacionales
  - Uso de NoSQL como fuente de datos
  - Interacción con datos basados en la web



# Datos de Ejemplo

- Uso de conjuntos de datos de juguete:
  - Incluidos en la biblioteca Scikit-learn.
  - Simples pero útiles para probar técnicas de manipulación de datos.
- Archivos recomendados:
  - Colores.txt
  - Titanic.csv
  - Values.xls
- Colocar los archivos en la carpeta correcta para evitar errores de entrada/salida (IO)

```
from google.colab import drive
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/Colab Notebooks/Colores.txt'
```

# Carga de Datos

- Carga en Memoria
  - Método más rápido y directo
  - Ideal para conjuntos de datos pequeños
- Muestra todos los datos del archivo al mismo tiempo

```
with open("Colores.txt", 'r') as open_file:  
    print('Contenido del archivo:\n' + open_file.read())
```

## Transmisión de Datos (*Streaming*)

- Procesa los datos por fragmentos en lugar de cargarlos completos
  - Cuando los datos son demasiado grandes
  - Útil para conjuntos de datos grandes o remotos
- Se leen los registros uno por uno

```
with open("Colores.txt", 'r') as open_file:  
    for observation in open_file:  
        print('Lectura de datos: ' + observation, end="")
```

# Muestreo de Datos

- Muestra solo registros seleccionados
  - Ahorra recursos al trabajar solo con lo necesario
  - Recupera datos en intervalos o de manera aleatoria
- Obtener cada 2do registro del archivo Colores.txt

```
n = 2
with open("Colores.txt", 'r') as open_file:
    for j, observation in enumerate(open_file):
        if j % n == 0:
            print('Línea: ' + str(j) + ' Contenido: ' + observation, end="")
```

# Muestreo de Datos Aleatorio en Archivos

- La salida aparece en orden numérico, pero las líneas seleccionadas son aleatorias
  - random() genera un valor entre 0 y 1
  - sample\_size define el porcentaje de líneas a seleccionar

```
from random import random
sample_size = 0.25
with open("Colores.txt", 'r') as open_file:
    for j, observation in enumerate(open_file):
        if random() <= sample_size:
            print('Reading Line: ' + str(j) + ' Content: ' + observation, end="")
```

```
Reading Line: 0 Content: Color      Value
Reading Line: 3 Content: Yellow    3
```

# Archivos Planos Estructurados

- Fáciles de leer y manipular.
- Tipos comunes: archivos de texto, CSV, Excel.
- Desventajas del manejo nativo en Python:
  - Los encabezados se tratan como datos regulares
  - No permite fácilmente seleccionar columnas específicas
- Para facilitar la manipulación de datos usar bibliotecas especializadas
  - Pandas

# Lectura de archivos de texto con pandas

- Interpreta correctamente encabezados
- Genera un índice automáticamente

```
import pandas as pd
color_table = pd.io.parsers.read_table("Colores.txt")
print(color_table)
```

	Color	Value
0	Red	1
1	Orange	2
2	Yellow	3
3	Green	4
4	Blue	5
5	Purple	6
6	Black	7
7	White	8

# Archivos CSV

- Campos separados por comas
- Registros separados por saltos de línea
- Cadenas entre comillas dobles

```
import pandas as pd
titanic = pd.io.parsers.read_csv("Titanic.csv")
X = titanic[['age']]
print(X)
```



# Lectura de Archivos Excel

- Soportan formatos complejos
- Contienen múltiples hojas

```
import pandas as pd
xls = pd.ExcelFile("Values.xls")
trig_values = xls.parse('Sheet1', index_col=None, na_values=['NA'])
print(trig_values)
```

## Resumen

- Los datos son esenciales para las aplicaciones
- Están dispersos, en distintos formatos y difíciles de interpretar
- Dominar técnicas para acceder en distintas ubicaciones
- Trabajar con datos en múltiples formatos
  - Texto plano: Simple, pero requiere conversión manual de tipos
  - CSV: Más formato, adecuado para datos tabulares
  - Excel: Soporte avanzado para múltiples hojas y formatos
- Herramientas recomendadas:
  - Pandas: Simplifica la lectura y manipulación



# Procesamiento de los datos

- Trabajar con NumPy y pandas
- Considerar el impacto de las fechas en los datos
- Reparar datos faltantes
- Cortar, combinar y modificar elementos de datos

## ¿Qué significa dar forma a los datos?

- La preparación adecuada garantiza análisis válidos y útiles
- Una buena preparación acelera el análisis y reduce errores
- La mayor parte del tiempo se dedica a:
  - Obtener y limpiar datos
  - Crear subconjuntos y combinarlos
  - Fusionar varios conjuntos en uno solo
- No cambiar los valores, solo reorganizarlos

# Problemas comunes al modelar datos

- Datos faltantes: Necesitan ser identificados y tratados
- Fechas: Pueden causar problemas si no se procesan adecuadamente
- Fusión de datos:
  - Los datos suelen provenir de diferentes fuentes
  - Es esencial unificar los datos en un formato estándar

# Herramientas clave: NumPy y pandas

- NumPy:
  - Ofrece alto rendimiento
  - Ideal para cálculos donde la velocidad es crucial
- pandas:
  - Simplifica tareas complejas como:
    - Series temporales
    - Estadísticas avanzadas
    - Funciones como groupby(), merge() y join()
  - Reduce el riesgo de errores

# Validación de los datos

- ¿Por qué validar?
  - Para asegurarse de que los datos son utilizables
  - Detectar y eliminar registros duplicados
- Limitación:
  - No garantiza que los datos sean 100% correctos, pero asegura que el análisis sea razonablemente válido

# Eliminación de Duplicados

- Los duplicados pueden eliminarse fácilmente con pandas

```
import pandas as pd
df = pd.DataFrame({'Number': [1, 2, 3, 3],
                   'String': ['First', 'Second', 'Third', 'Third'],
                   'Boolean': [True, False, True, True]})

print(df.drop_duplicates())
```



# Manejo de Fechas en Datos

- Problemas comunes:
- Representación numérica varía según la plataforma o usuario
  - Zonas horarias y formato de tiempo
- Formateo de Fechas y Horas
  - Usando str() y strftime().

i

```
import datetime as dt
now = dt.datetime.now()
print(str(now)) # Resultado en formato predeterminado
print(now.strftime('%a, %d %B %Y')) # Formato personalizado
```

# Transformaciones de Tiempo

- Cambiar zonas horarias o calcular diferencias horarias

```
from datetime import datetime, timedelta
now = datetime.now()
future_time = now + timedelta(hours=2)
print(f"Ahora: {now.strftime('%H:%M:%S')}")
print(f"Futuro: {future_time.strftime('%H:%M:%S')}")
```

# Manejo de Datos Faltantes

- Identificar Datos Faltantes
  - Representaciones comunes: np.NaN, None.

```
import pandas as pd
import numpy as np
s = pd.Series([1, 2, 3, np.NaN, 5, None])
print(s.isnull()) # Indica qué valores están faltando.
```

# Estrategias para Datos Faltantes

1. Ignorar: No recomendado; sesga los análisis.
2. Rellenar valores faltantes (fillna)
  - Usar la media, mediana, etc.
3. Eliminar valores faltantes (dropna)

```
s = pd.Series([1, 2, 3, np.NaN, 5, None])  
print(s.fillna(s.mean())) # Rellenar con la media  
print(s.dropna()) # Eliminar valores faltantes
```

# Imputación de Datos Faltantes

- Usar SimpleImputer para completar datos faltantes.
- Métodos disponibles:
  - Media (mean).
  - Mediana (median).
  - Valor más frecuente (most\_frequent).

```
from sklearn.impute import SimpleImputer
s = pd.DataFrame([1, 2, 3, np.nan, 5, 6, np.nan])
imp = SimpleImputer(strategy='mean', add_indicator=True)
x = imp.fit_transform(s)
print(x)
```

# Filtrado y selección de datos

- División y Segmentación
- Filtrado de datos para mejorar la precisión y eficiencia del análisis
  - Trabajar con un subconjunto específico de datos según las necesidades
    - Analizar solo una columna como "edad"
    - Filas específicas (personas entre 5 y 10 años).
- Pasos clave:
  1. Filtrar filas según criterios seleccionados.
  2. Seleccionar columnas relevantes.

## Segmentación de filas

- Análisis de datos en un momento específico
- Filtrar datos de una matriz 2D o 3D por filas específicas.

```
x = np.array( [[[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
               [[11,12,13], [14,15,16], [17,18,19]],  
               [[21,22,23], [24,25,26], [27,28,29]]])  
  
x[1]
```

## Corte de columnas

- Observar patrones en un producto o ubicación específica
- Seleccionar datos en un ángulo perpendicular a las filas (columnas).

```
x = np.array( [[[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
               [[11,12,13], [14,15,16], [17,18,19]],  
               [[21,22,23], [24,25,26], [27,28,29]])  
  
x[:,1]
```



## División combinada (Dicing)

- Obtención de segmentos específicos de datos
- Cortar filas y columnas simultáneamente para obtener una cuña de datos

```
x = np.array( [[[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
              [[11,12,13], [14,15,16], [17,18,19]],  
              [[21,22,23], [24,25,26], [27,28,29]]])  
  
print(x[1,1])  
print(x[:,1,1])  
print(x[1,:,1])  
print(f"\n{x[1:2, 1:2]}")
```

# Concatenación y transformación

- Evitar datos dispersos en múltiples bases de datos con formatos diferentes
- Combinar datos en un único conjunto para análisis.

```
import pandas as pd
df = pd.DataFrame({'A': [2,3,1],
                   'B': [1,2,3],
                   'C': [5,3,4]})

df1 = pd.DataFrame({'A': [4], 'B': [4], 'C': [4]})
df = pd.concat([df, df1])
df = df.reset_index(drop=True)
```

# Eliminación de datos

- Datos más limpios y enfocados
- Simplificar datos eliminando casos o columnas innecesarias

```
import pandas as pd
df = pd.DataFrame({'A': [2,3,1],
                   'B': [1,2,3],
                   'C': [5,3,4]})

df = df.drop(df.index[[1]])
df = df.drop(columns=['B'])
```

# Ordenación y mezcla

- Ordenación:
  - Presentación de datos en un orden específico.
  - Uso: `sort_values()` y `reset_index()`.
- Mezcla:
  - Romper patrones en los datos.
  - Uso: `np.random.shuffle()` para reordenar índices.

```
df = pd.DataFrame({'A': [2,1,2,3,3,5,4],  
                  'B': [1,2,3,5,4,2,5],  
                  'C': [5,3,4,1,1,2,3]})  
  
df = df.sort_values(by=['A', 'B'], ascending=[True, True])
```

## Agregación de datos

- Datos agrupados y transformados para facilitar el análisis.
- Combinar o agrupar datos usando funciones como promedio, suma o varianza.

```
df = pd.DataFrame({'Map': [0,0,0,1,1,2,2],  
                  'Values': [1,2,3,5,4,2,5]})  
  
df['S'] = df.groupby('Map')['Values'].transform(np.sum)  
df['M'] = df.groupby('Map')['Values'].transform(np.mean)  
df['V'] = df.groupby('Map')['Values'].transform(np.var)
```

# **Unidad 5:** **Visualizar información**



## Ejemplos de Visualización de gráficos

- Para obtener información sobre la visualización de datos
  - [https://pandas.pydata.org/pandas-docs/dev/user\\_guide/visualization.html](https://pandas.pydata.org/pandas-docs/dev/user_guide/visualization.html)

**CampusDual TIC**

Gracias por su atención

