



PYTHON

LA HECHICERA DEL CÓDIGO



Dream

PYTHON



PRINT

Es una declaración que al ejecutarse muestra o (imprime), en pantalla el argumento que se introduce dentro de los paréntesis ().

Para mostrar el texto ingresamos entre comillas ya sean dobles o simples (" "), los caracteres de texto que deben mostrarse en pantalla.

```
print ("Hola")  
>>Hola
```

STRING

Los strings en Python son un tipo de dato formado por cadenas o **secuencias** de caracteres de cualquier tipo, formando un texto.

Es la unificación de cadenas de texto:

```
print('Hello + " " + Kitty ')  
>>Hello Kitty
```

CONCATENACIÓN

Caracteres especiales

Le indicamos a la consola el carácter a continuación del símbolo \ debe ser tratado como un carácter especial.

- \ " > Imprime comillas
- \ n > Separa el texto en una nueva línea
- \ t > Imprime un tabulador
- \\ > Imprime la barra invertida textualmente.



PYTHON

LA HECHICERA DEL CÓDIGO

Python es un lenguaje de programación de alto nivel que se destaca por ser fácil de aprender, legible y versátil. Se utiliza en áreas muy diversas como desarrollo web, análisis de datos, inteligencia artificial, automatización de tareas y más.

Python se caracteriza por:

- Sencillez y claridad: Su sintaxis es cercana al lenguaje humano, lo que facilita su aprendizaje.
- Multiparadigma: Permite programar de manera estructurada, orientada a objetos o funcional.
- Amplia comunidad: Tiene muchas librerías y recursos disponibles, lo que acelera el desarrollo de proyectos.

Isela Data Maven



PYTHON

La hechicera del código



Input:

Es la función que permite al usuario introducir información por medio del teclado al ejecutarse, otorgándole una instrucción acerca del ingreso solicitado. El código continuará ejecutándose luego que el usuario realice la acción.

```
input("Escribe tu nombre : " )
```

```
>> Escribe tu nombre : |
```

```
print("Tu nombre es" + input("Escribe tu nombre : " )
```

```
>>Escribe tu nombre: La hechicera del código
```

```
>> Tú nombre es La hechicera del código
```

En Python los tipos de datos son las diferentes clases de valores que puede manejar Python.

Algunos de los más comunes son:

- **int** → números enteros (ej. 10, -5, 2025)
- **float** → números decimales (ej. 3.14, -0.5)
- **str** → cadenas de texto (ej. "Hola", 'Python')
- **bool** → valores lógicos (ej. True, False)

Ejemplo:

- `print(type(10))` **# int**
- `print(type(3.14))` **# float**
- `print(type("Hola"))` **# str**
- `print(type(True))` **# bool**



Variables

Una variable es un espacio en memoria donde guardamos datos.

En Python no es necesario indicar el tipo de dato al declarar la variable, ya que se asigna automáticamente.

Ejemplo:

- `nombre = "Isela"`
- `edad = 25`
- `pi = 3.1416`
- `es_estudiante = True`
- `print(nombre, edad, pi, es_estudiante)`

Nombres de Variables y Reglas

✓ Reglas:

1. Deben comenzar con una letra o guion bajo (`_`), nunca con un número.
2. Pueden contener letras, números y guion bajo, pero no espacios.
3. No pueden ser palabras reservadas de Python (ej. `if`, `for`, `while`).
4. Python distingue mayúsculas y minúsculas (`Edad` \neq `edad`).

Ejemplo:

- `nombre_usuario = "Ana"`
- `Edad = 30`
- `_pi = 3.14`



✗ Ejemplos inválidos

- `2edad = 25` # No puede empezar con número
- `nombre-usuario = "Ana"` # No se permiten guiones
- `for = 5` # "for" es palabra reservada

Integers (int)

Son números enteros sin decimales.

Ejemplo:

- `x = 10`
- `y = -5`
- `print(x + y)` # 5
- `print(type(x))` # <class 'int'>

Float

Son números con decimales o en notación científica.

Ejemplo:

- `a = 3.5`
- `b = -0.7`
- `c = 1.2e3` # Notación científica = 1200.0
- `print(a + b)` # 2.8
- `print(c)` # 1200.0
- `print(type(a))` # <class 'float'>



Conversiones entre Tipos de Datos

Se pueden transformar datos usando funciones de conversión:

- **int()** → convierte a entero
- **float()** → convierte a decimal
- **str()** → convierte a cadena

Ejemplo:

```
x = 10  
y = 3.5
```

```
print(float(x)) # 10.0  
print(int(y))   # 3  
print(str(x))   # "10"
```

Formatear Cadenas con format()

Permite insertar valores en una cadena de texto.

Ejemplo:

```
nombre = "Isela"  
edad = 25  
print("Hola, me llamo {} y tengo {}  
años".format(nombre, edad))
```



Ejemplo con posición y formato numérico:

```
pi = 3.14159265
```

```
print("El valor de PI con 2 decimales:{:.2f}".format(pi))
```

Operadores Matemáticos

Python permite realizar operaciones matemáticas con símbolos:

Operador	Significado	Ejemplo	Resultado
+	Suma	5 + 3	8
-	Resta	(7 - 2)	6
*	Multiplicación	2 * 6	12
/	División	(7/2)	3.5
//	División entera	7 // 2	3
%	Módulo (residuo)	7 % 2	1
**	Potencia	2 ** 3	8

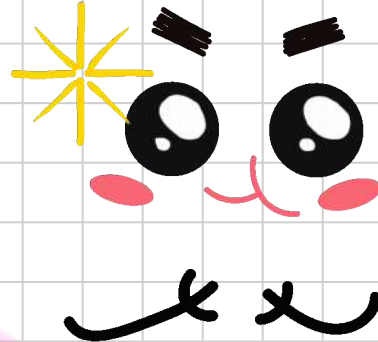


Ejemplo



a = 7
b = 2

```
print(a + b) # 9  
print(a - b) # 5  
print(a * b) # 14  
print(a / b) # 3.5  
print(a // b) # 3  
print(a % b) # 1  
print(a ** b) # 49
```



Conversión de tipos (int(), float(), str())

TIPOS DE DATOS	Significado
Números Enteros (int)	Son números sin punto decimal. Como cuando cuentas manzanas: 1, 5, 100, etc.
Números Decimales (float):	Son números con punto decimal. Como cuando mides algo: 3.14, 2.5, 99.9.
Cadenas de Texto (str):	Son palabras, letras, símbolos o cualquier cosa que no sea un número. Siempre van entre comillas. Por ejemplo: "hola", "Manzana", "123".

Ahora, la Conversión de Tipos es el proceso de cambiar lo que está en un recipiente para meterlo en otro. En Python, usamos estas "fórmulas mágicas" para hacerlo:



1. `int()`: Para convertir a un número entero

Imagina que tienes el número "25" guardado como texto. Si quieres hacer una operación matemática, como sumarle 5, Python no te dejará porque ve un texto, no un número. Con `int()` lo solucionas:

- Lo que tienes: "25" (es un texto)
- Lo que haces: `int("25")`
- Lo que obtienes: 25 (ahora es un número entero con el que puedes operar).

2. `float()`: Para convertir a un número decimal

Si necesitas trabajar con precios o medidas, `float()` es tu aliado.

- Lo que tienes: 20 (es un número entero)
- Lo que haces: `float(20)`
- Lo que obtienes: 20.0 (ahora es un número decimal).

También funciona si lo que tienes es texto:

- `float("3.14")` te da 3.14.

3. `str()`: Para convertir a texto

Esta es útil cuando quieres combinar un número con un texto. Por ejemplo, si quieres decir "Tu edad es 25".

- Lo que tienes:
 - "Tu edad es " (es un texto)
 - 25 (es un número)
- Lo que haces: `"Tu edad es " + str(25)`
- Lo que obtienes: "Tu edad es 25" (un solo texto completo).



Redondeo de números (round())

Imagina que tienes un número con muchos decimales, como si fuera un precio muy largo: 15.78345. A veces, para que sea más fácil de entender o usar, necesitas "simplificarlo" o "recortarlo".

Ahí es donde entra la función `round()`. Piensa en ella como una tijera mágica que redondea los números.

¿Cómo funciona `round()`?

La forma más básica es para redondear a un número entero:

- Si el primer decimal es 5 o más, redondea hacia arriba.
 - Por ejemplo, `round(15.7)` se convierte en 16.
 - 15.7 está más cerca de 16 que de 15.
- Si el primer decimal es menos de 5, redondea hacia abajo.
 - Por ejemplo, `round(15.3)` se convierte en 15.
 - 15.3 está más cerca de 15 que de 16.

¿Y si quiero redondear a 2 decimales?

Puedes decirle a `round()` exactamente cuántos decimales quieres. Para eso, le das dos datos:

- 1.El número que quieres redondear.
- 2.La cantidad de decimales que deseas.

Ejemplo: Imagina que tienes el número 3.14159. Si quieres redondearlo a 2 decimales, le dices a Python:

- `round(3.14159, 2)`

El resultado sería 3.14. La función se fija en el tercer decimal (1) y, como es menor que 5, no sube el número anterior.

En resumen:

- `round(número)` → Lo simplifica al entero más cercano.
- `round(número, decimales)` → Lo simplifica a la cantidad de decimales que tú le digas.

Así de sencillo es usar `round()` para que tus números se vean más ordenados.



Formateo de cadenas (format, f-strings)

Imagina que quieres escribir una carta o un mensaje. A veces, necesitas llenar "espacios en blanco" con información que cambia. Por ejemplo, "Hola, mi nombre es [nombre] y tengo [edad] años".

En Python, el formateo de cadenas es la manera elegante y poderosa de llenar esos espacios vacíos. Hay dos formas principales de hacerlo, y son muy sencillas:

1. El método `.format()` (La forma clásica)

Este método funciona como un rompecabezas. Creas una plantilla con {} para los espacios vacíos y luego, con `.format()`, le pasas las piezas (los datos) en orden. Ejemplo:

```
Untitled - TextEdit
File Edit View Help PYTHON

nombre = "María"
edad = 25
#Usando .format()
mensaje = "Hola, mi nombre es {} y tengo {} años.".format(nombre, edad)

print(mensaje) # Salida: Hola, mi nombre es María y tengo 25 años.
```




Aquí, Python reemplaza el primer {} con el primer dato que le das (nombre) y el segundo {} con el segundo dato (edad). Es muy organizado.

2. Las f-strings (La forma moderna y más fácil)

Las f-strings (la "f" es de formatted) son la forma más popular y sencilla de hacerlo hoy en día. Solo tienes que poner una f al principio de tu cadena de texto y luego, dentro de la cadena, metes las variables directamente entre llaves {}.

Ejemplo:

```
nombre = "Carlos"
edad = 30
# Usando f-strings
mensaje = f"Hola, mi nombre es {nombre} y
tengo {edad} años."
print(mensaje) # Salida: Hola, mi nombre es
Carlos y tengo 30 años.
```



¿Por qué son tan populares las f-strings?

- Más legibles: Puedes ver la cadena completa con las variables en su lugar, lo que hace el código más claro.
- Más cortas: Te ahorras el `.format()` y escribir los nombres de las variables dos veces.
- Más rápidas: Técnicamente, son un poco más rápidas, aunque para la mayoría de los casos no notarás la diferencia.

En resumen:

- `.format()` es el método tradicional, funciona como un rompecabezas.
- f-strings son el método moderno, te permiten meter las variables directamente en la cadena con una `f` al inicio.

Ambos hacen lo mismo: crear cadenas de texto personalizadas y dinámicas de manera sencilla.

LA HECHICERA
DEL CÓDIGO



2. Strings (Cadenas de texto)

Métodos básicos:

Imagina una cadena de texto como un tren. Cada vagón es una letra o un carácter. Para trabajar con este tren, Python tiene dos herramientas principales:

1. `.index()`: Encontrar un vagón específico

El método `.index()` es como un inspector que te ayuda a encontrar la posición de un vagón en particular. La posición en Python siempre empieza a contar desde 0, no desde 1.

- ¿Qué hace? Busca la primera vez que aparece un carácter o una secuencia de caracteres y te dice en qué posición está.
- Ejemplo:
 - Si tienes la palabra "Hola"
 - `"Hola".index("o")` te dirá 1, porque la "o" está en la posición 1.
 - `"Hola".index("la")` te dirá 2, porque la secuencia "la" empieza en la posición 2.

2. Slicing (`text[0:5]`): Cortar un pedazo del tren

El slicing es como un tren de juguete que puedes cortar para quedarte solo con los vagones que necesitas. Usas corchetes `[]` y un rango de números para decirle a Python desde dónde y hasta dónde quieres cortar.

La sintaxis es `[inicio:fin]`, pero hay una regla importante: el número de fin no se incluye. Siempre se corta hasta el vagón anterior.

- Ejemplo:
 - Si tienes la palabra "Python"
 - `"Python"[0:2]` te dará "Py". (Empieza en la posición 0 y corta hasta la 2, sin incluirla).
 - `"Python"[2:5]` te dará "tho". (Empieza en la posición 2 y corta hasta la 5, sin incluirla).
 - También puedes dejar el inicio o el fin vacíos:
 - `"Python"[:2]` te da "Py". (Si no pones inicio, empieza desde 0).
 - `"Python"[2:]` te da "thon". (Si no pones fin, corta hasta el final).

En resumen:

- `.index()` te da la posición de algo que buscas.
- Slicing te permite extraer un pedazo de la cadena usando posiciones.

Estas dos herramientas son esenciales para manipular texto y sacar la información que te interesa de una cadena.



Métodos de análisis:

Imagina que cada uno de estos métodos es una pregunta que le haces a una cadena de texto para saber qué tipo de caracteres tiene. La respuesta siempre será True (verdadero) o False (falso).

Para saber si son números:

- **isdigit()**: Le preguntas a la cadena: "¿Estás hecha de dígitos (0-9)?"
 - "123".isdigit() te responderá True.
 - "12.3".isdigit() te responderá False (por el punto decimal).
- **isnumeric()**: Es un poco más flexible. Le preguntas: "¿Puedes ser un número, aunque no seas un simple dígito?"
 - "123".isnumeric() te responde True.
 - "1/2".isnumeric() te responde True (la fracción 1/2 se considera un valor numérico).
- **isdecimal()**: Es la más estricta de las tres. Le preguntas: "¿Eres un número decimal del sistema arábigo (0-9)?"
 - "123".isdecimal() te responde True.
 - "1/2".isdecimal() te responde False.

Para saber si son letras o números:

- **isalpha()**: Le preguntas a la cadena: "¿Estás hecha de solo letras?"
 - "Hola".isalpha() te responderá True.
 - "Hola123".isalpha() te responderá False.
- **isalnum()**: Le preguntas: "¿Estás hecha de letras y/o números?" (Alfanumérico)
 - "Hola123".isalnum() te responderá True.
 - "Hola 123".isalnum() te responderá False (por el espacio).

Para saber si son mayúsculas o minúsculas:

- **islower()**: Le preguntas: "¿Todas tus letras son minúsculas?"
 - "hola".islower() te responderá True.
 - "Hola".islower() te responderá False.
- **isupper()**: Le preguntas: "¿Todas tus letras son mayúsculas?"
 - "HOLA".isupper() te responderá True.
 - "Hola".isupper() te responderá False.

Para saber si es un espacio o si se puede imprimir:

- **isspace()**: Le preguntas: "¿Eres solo un espacio en blanco, un salto de línea o una tabulación?"
 - " ".isspace() te responderá True.
 - " hola ".isspace() te responderá False.
- **isprintable()**: Le preguntas: "¿Todos tus caracteres se pueden ver en pantalla?"
 - "Hola mundo".isprintable() te responderá True.
 - Una cadena con caracteres invisibles (como un salto de línea) te responderá False.



Métodos de transformación:

Imagina que una cadena de texto es una pieza de arcilla. Los métodos que vamos a ver son herramientas que te permiten moldear, cambiar y limpiar esa pieza.

1. Cambio de mayúsculas y minúsculas

Estos métodos son como interruptores para el estilo de las letras:

- `cadena.capitalize()`: Pone la primera letra en mayúscula y el resto en minúsculas.
 - Ejemplo: "hola mundo".capitalize() se convierte en "Hola mundo".
- `cadena.lower()`: Convierte toda la cadena a minúsculas.
 - Ejemplo: "Hola Mundo".lower() se convierte en "hola mundo".
- `cadena.upper()`: Convierte toda la cadena a mayúsculas.
 - Ejemplo: "Hola Mundo".upper() se convierte en "HOLA MUNDO".
- `cadena.swapcase()`: Invierte el estilo. Las mayúsculas se vuelven minúsculas y viceversa.
 - Ejemplo: "Hola Mundo".swapcase() se convierte en "hOLA mUNDO".

2. Limpieza de espacios

Estos métodos son como borradores que eliminan los espacios innecesarios.

- `cadena.strip()`: Quita los espacios en blanco del principio y del final de la cadena.
 - Ejemplo: " hola mundo ".strip() se convierte en "hola mundo".
- `cadena.lstrip()`: Quita los espacios solo del lado izquierdo (del principio).
 - Ejemplo: " hola mundo ".lstrip() se convierte en "hola mundo".
- `cadena.rstrip()`: Quita los espacios solo del lado derecho (del final).
 - Ejemplo: " hola mundo ".rstrip() se convierte en " hola mundo".

3. Formato y alineación

Estos métodos te permiten alinear el texto de forma estética. Debes indicar el número total de espacios que quieres que ocupe la cadena.

- `cadena.center(ancho)`: Centra el texto en un espacio dado, rellenando con espacios a los lados.
 - Ejemplo: "hola".center(10) se convierte en " hola ".
- `cadena.ljust(ancho)`: Alinea el texto a la izquierda.
 - Ejemplo: "hola".ljust(10) se convierte en "hola ".
- `cadena.rjust(ancho)`: Alinea el texto a la derecha.
 - Ejemplo: "hola".rjust(10) se convierte en " hola ".

4. Otros métodos importantes

- `cadena.replace(viejo, nuevo)`: Busca una parte de la cadena y la reemplaza por otra.
 - Ejemplo: "Hola, mundo".replace("mundo", "amigo") se convierte en "Hola, amigo".
- `cadena.encode()`: Convierte la cadena en bytes, que es el formato que las computadoras entienden. Por lo general, se usa con un tipo de codificación como utf-8.
 - Ejemplo: "Hola".encode("utf-8") se convierte en b'Hola'. Esto es un tema más avanzado, pero es importante saber que existe.



Métodos de separación y unión:

Imagina que una cadena de texto es como una soga. Estos métodos son herramientas para cortar o unir esa soga de diferentes maneras.

Métodos de Separación (para cortar la soga)

Estos métodos toman una cadena y la dividen en varias partes, guardándolas en una lista.

- **cadena.split(separador):** El método más común para separar. Corta la cadena en pedazos usando un separador que tú le indiques. Si no le das uno, usa los espacios por defecto.
 - Ejemplo: frase = "Hola, mundo de Python"
 - frase.split(" ") te da ['Hola,', 'mundo', 'de', 'Python']
 - frase.split(",") te da ['Hola', ' mundo de Python']
- **cadena.splitlines():** Corta la soga por las líneas. Divide la cadena en una lista de líneas, usando los saltos de línea (como cuando presionas "Enter").
 - Ejemplo: texto = "Línea 1\nLínea 2\nLínea 3"
 - texto.splitlines() te da ['Línea 1', 'Línea 2', 'Línea 3']
- **cadena.partition(separador):** Corta la soga en tres partes. Busca el primer separador que le des y divide la cadena en una tupla de 3 elementos:
 - a. La parte antes del separador.
 - b. El separador mismo.
 - c. La parte después del separador.
 - Ejemplo: url = "www.google.com"
 - url.partition(".") te da ('www', '.', 'google.com')
- **cadena.rpartition(separador):** Corta la soga en tres, pero desde la derecha. Es igual que partition(), pero busca la última vez que aparece el separador.
 - Ejemplo: ruta = "C:/usuarios/documentos/archivo.txt"
 - ruta.rpartition("/") te da ('C:/usuarios/documentos', '/', 'archivo.txt')

Método de Unión (para pegar pedazos de soga)

- **separador.join(lista):** El método para unir. Es el inverso de split(). Toma una lista de cadenas y las une en una sola, colocando el separador entre cada una.
 - Ejemplo: palabras = ['Hola', 'mundo', 'de', 'Python']
 - ".join(palabras) te da 'Hola mundo de Python'
 - "-".join(palabras) te da 'Hola-mundo-de-Python'



3. Estructuras de Datos

Listas → colección ordenada y mutable

Imagina una lista como una lista de compras 🛒 que haces en un papel.

Colección Ordenada

"Colección ordenada" significa que cada cosa que pones en tu lista tiene una posición específica y esa posición importa. Al igual que en una lista de compras:

- El primer artículo está en la posición 1.
- El segundo artículo está en la posición 2.

En Python, la computadora cuenta las posiciones, pero empieza desde el cero (0). Así que el primer artículo está en el índice 0, el segundo en el 1, y así sucesivamente.

Ejemplo:

```
lista_de_compras = ["manzanas", "leche",  
"pan"]
```



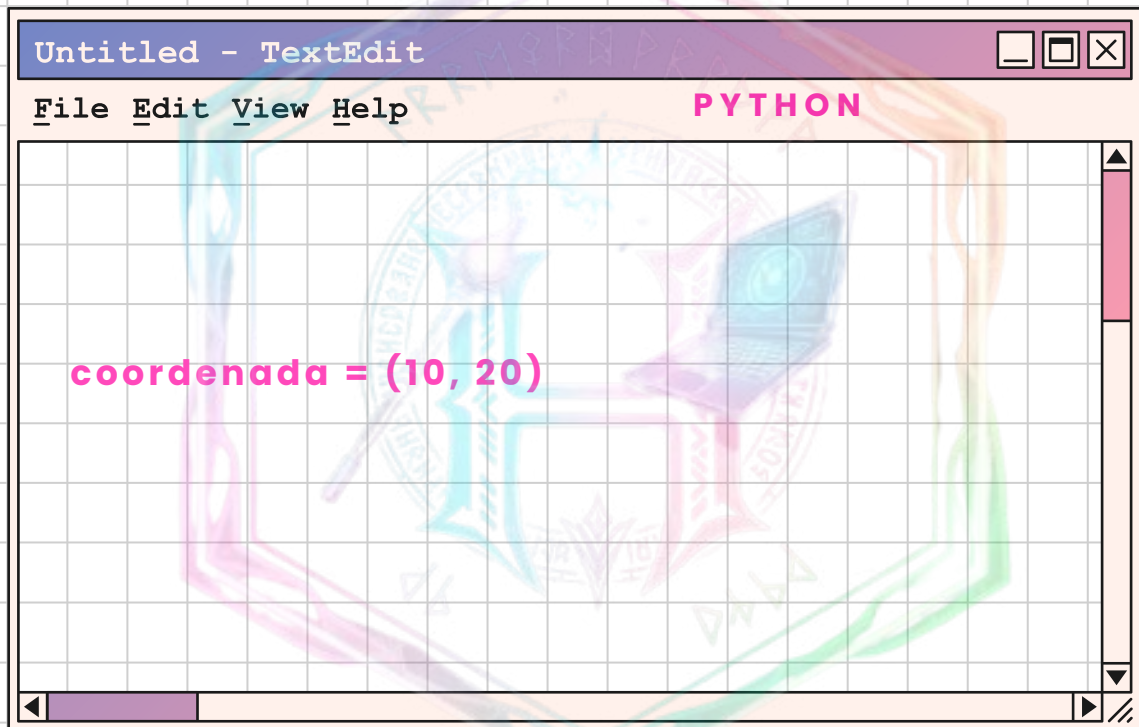
Tuplas → colección ordenada e inmutable

Las tuplas en Python son como una lista de compras escrita en una piedra. 🪄

Colección Ordenada

Al igual que las listas, las tuplas son una colección ordenada de elementos. Esto significa que cada elemento tiene una posición fija (empezando desde el índice 0), y el orden en el que se guardan es importante.

Ejemplo:





Tuplas → colección ordenada e inmutable

- El número 10 está en la posición 0.
- El número 20 está en la posición 1.

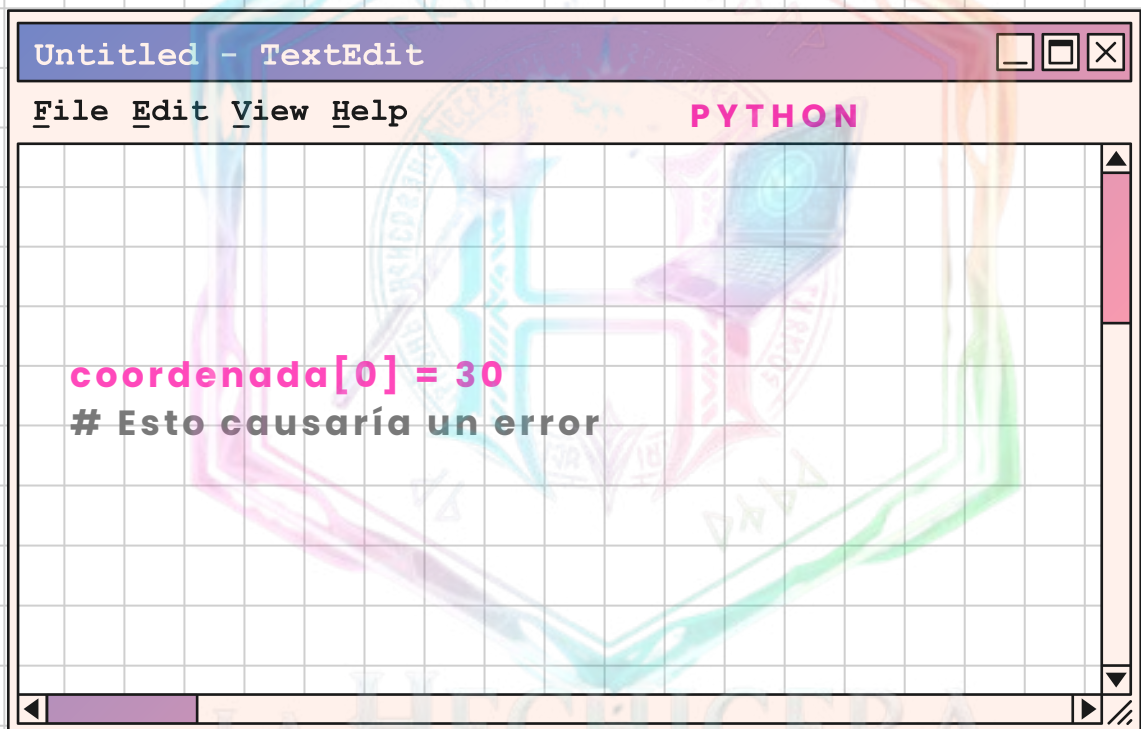
Colección Inmutable

"Immutable" significa que no pueden ser cambiadas una vez que han sido creadas. Una tupla es fija.

Esto implica que no puedes:

- Añadir nuevos elementos.
- Quitar elementos.
- Cambiar un elemento por otro.

Ejemplo: Si intentaras cambiar un elemento de la tupla coordenada, Python te daría un error:



¿Para qué se usan las tuplas?

Se usan para guardar datos que no deben ser modificados, como:

Las coordenadas de un punto ((x, y)).

Los días de la semana.

La información de un registro que no va a cambiar (por ejemplo, los datos de un libro: ("El Señor de los Anillos", "J.R.R. Tolkien", 1954)).

En resumen, la principal diferencia entre una lista y una tupla es que las listas son flexibles (se pueden editar) y las tuplas son rígidas (son permanentes). Esto las hace perfectas para guardar datos que necesitan mantenerse intactos.



Diccionarios → pares clave-valor

Imagina un diccionario como una agenda de contactos. No organizas a tus amigos por un número de lista (como en una lista), sino por su nombre. Cada entrada tiene dos partes:

1. **La clave (key):** El nombre de la persona, que es única e irrepetible.
2. **El valor (value):** La información asociada a esa persona (su teléfono, correo, dirección, etc.).

Los diccionarios en Python funcionan exactamente igual: son colecciones que guardan información en pares de clave-valor. La clave es como el "rótulo" o la "etiqueta" que te permite encontrar el valor que necesitas, de forma rápida y directa.

¿Cómo se ven en Python?

Los diccionarios se crean con llaves {}. Dentro de ellas, cada par se separa por dos puntos :, y cada par de clave-valor está separado por una coma ,.

Ejemplo:

```
Untitled - TextEdit
File Edit View Help PYTHON
perfil_usuario = {
    "nombre": "Simba",
    "edad": 1,
    "ciudad": "Ciudad de México"
}
```



PYTHON

La hechicera del código



- Claves: "nombre", "edad", "ciudad"
- Valores: "Ana", 30, "Ciudad de México"

¿Cómo se usan?

Para acceder a la información, no usas la posición (como en las listas), sino la clave.

- Para saber el nombre de Ana, simplemente usas la clave "nombre":

```
print(perfil_usuario["nombre"])

# Salida: Ana
```

Además, los diccionarios son mutables (pueden ser modificados). Puedes cambiar un valor o añadir uno nuevo fácilmente:

LA HECHICERA
DEL CÓDIGO



```
Untitled - TextEdit
File Edit View Help PYTHON

perfil_usuario["edad"] = 31
# Cambia la edad a 31
perfil_usuario["profesion"] =
"Ingeniera"
# Añade una nueva clave-valor
```

En resumen, un diccionario es ideal para cuando necesitas guardar información que tiene una etiqueta o nombre para ser encontrada, en lugar de una simple posición.

Sets → **colección no ordenada de elementos únicos**

Métodos: `add()`, `clear()`, `copy()`, `difference()`, `difference_update()`, `discard()`, `intersection()`, `intersection_update()`, `isdisjoint()`, `issubset()`, `pop()`, `remove()`, `symmetric_difference()`, `symmetric_difference_update()`, `union()`, `update()`



PYTHON

La hechicera del código



Método	Descripción Simple	Ejemplo
add()	Añade un elemento a un set. Si ya existe, no hace nada.	<code>mi_set.add('plátano')</code>
clear()	Elimina todos los elementos del set.	<code>mi_set.clear()</code>
copy()	Crea una copia superficial del set.	<code>mi_nueva_copia = mi_set.copy()</code>
difference()	Devuelve un nuevo set con los elementos que están en el primer set, pero no en el segundo.	<code>set1.difference(set2)</code>
difference_update()	Elimina del set los elementos que también están en otro set.	<code>set1.difference_update(set2)</code>
discard()	Elimina un elemento del set. No da error si el elemento no existe.	<code>mi_set.discard('manzana')</code>
intersection()	Devuelve un nuevo set con los elementos que tienen en común dos o más sets.	<code>set1.intersection(set2)</code>
intersection_update()	Mantiene solo los elementos que el set tiene en común con otro.	<code>set1.intersection_update(set2)</code>
isdisjoint()	Devuelve True si los sets no tienen elementos en común.	<code>set1.isdisjoint(set2)</code>
issubset()	Devuelve True si todos los elementos de un set están en otro.	<code>set1.issubset(set2)</code>
pop()	Elimina y devuelve un elemento aleatorio del set.	<code>elemento = mi_set.pop()</code>
remove()	Elimina un elemento del set. Da un error si no existe.	<code>mi_set.remove('naranja')</code>
symmetric_difference()	Devuelve un set con los elementos que están en uno u otro, pero no en ambos.	<code>set1.symmetric_difference(set2)</code>
symmetric_difference_update()	Mantiene solo los elementos que no están en ambos sets.	<code>set1.symmetric_difference_update(set2)</code>
union()	Devuelve un nuevo set con todos los elementos de ambos sets.	<code>set1.union(set2)</code>
update()	Añade los elementos de otro set (o cualquier iterable) al set actual.	<code>mi_set.update(otro_set)</code>



4. Booleanos y Control de Flujo

Booleanos: El interruptor ON/OFF de la lógica

Imagina un interruptor de luz. Solo tiene dos estados posibles:

- Encendido (ON)
- Apagado (OFF)

En Python, los Booleanos funcionan igual. Son un tipo de dato que solo puede tener dos valores:

- True (Verdadero) → Como el interruptor ON
- False (Falso) → Como el interruptor OFF



¿Para qué sirven?

Se usan para tomar decisiones. Cuando haces una pregunta a Python (una condición), la respuesta siempre es True o False.

Ejemplo de la vida real:

- Pregunta: "¿Está lloviendo?"
 - Respuesta: True (sí, está lloviendo)
 - Respuesta: False (no, no está lloviendo)

Ejemplo en Python:

```
Untitled - TextEdit
File Edit View Help PYTHON

edad = 18

# Pregunta a Python: ¿La edad es
# mayor o igual a 18?
es_mayor_de_edad = (edad >= 18)

print(es_mayor_de_edad)
# Salida: True
```



PYTHON

La hechicera del código



Si edad fuera 16, la respuesta sería False.

En resumen: Los booleanos son la base de la toma de decisiones en la programación. Son un simple "sí" o "no", "verdadero" o "falso".

Operadores de comparación (==, !=, <, >, <=, >=)

Aquí tienes una explicación sencilla de los operadores de comparación en Python, con un ejemplo visual para cada uno.

Operadores de Comparación: Hacer Preguntas y Obtener Respuestas

Imagina que los operadores de comparación son como un detective que te ayuda a hacer preguntas para comparar dos cosas. La respuesta a cada pregunta siempre es un booleano: True o False.

Operador	Significado	Lógica	Ejemplo	Resultado
and	Y	Se usa para combinar dos condiciones. La respuesta es True solo si ambas condiciones son True.	(2 > 1) and (5 > 3)	VERDADERO
or	O	Se usa para combinar dos condiciones. La respuesta es True si al menos una de las condiciones es True.	(10 < 5) or (10 > 8)	VERDADERO
not	NO	Invierte el resultado de una condición. not True se convierte en False, y not False se convierte en True.	not (2 == 3)	VERDADERO



PYTHON

La hechicera del código



Operadores Lógicos: Conectores para Tomar Decisiones

Operador	Significado	Pregunta que hace	Ejemplo	Resultado
(==)	Igual a	¿Son los dos valores iguales?	5 == 5	VERDADERO
!=	Diferente a	¿Son los dos valores diferentes?	10 != 5	VERDADERO
<	Menor que	¿El valor de la izquierda es menor que el de la derecha?	3 < 8	VERDADERO
>	Mayor que	¿El valor de la izquierda es mayor que el de la derecha?	12 > 10	VERDADERO
<=	Menor o igual que	¿El valor de la izquierda es menor o igual que el de la derecha?	4 <= 4	VERDADERO
>=	Mayor o igual que	¿El valor de la izquierda es mayor o igual que el de la derecha?	7 >= 5	VERDADERO



Ejemplo de la vida real:

Imagina que quieres ir al cine. Las condiciones son:

- **Condición 1:** `tienes_dinero = True`
- **Condición 2:** `la_pelicula_esta_disponible = True`
- **Con and:** Solo vas al cine si tienes dinero y la película está disponible.
 - `tienes_dinero and la_pelicula_esta_disponible → True`
- **Con or:** Vas al cine si tienes dinero o la película está disponible (aunque la lógica no sea perfecta, la computadora lo evaluaría).
 - `tienes_dinero or la_pelicula_esta_disponible → True`
- **Con not:** Si tu amigo te dice "no vamos a ver esa película", estás invirtiendo la condición original.
 - `not la_pelicula_esta_disponible → False` (si originalmente era True)

LA HECHICERA
DEL CÓDIGO



Condicionales (if, elif, else)

¡Claro que sí! Con gusto te explico los condicionales (if, elif, else) de la manera más sencilla.

Imagina que estás en la vida real y necesitas tomar decisiones. Siempre usas preguntas y, dependiendo de la respuesta, haces algo diferente.

if, elif, else: El Cerebro de tus Decisiones

Estos son los bloques de construcción para que tus programas tomen decisiones.

1. if (SI... sucede esto)

Es la condición principal. Le preguntas algo al programa y, Si esa condición es True, el programa ejecuta un bloque de código.

- Ejemplo: if hace_sol: (Si hace sol...) entonces voy a la playa.

2. elif (SINO, SI... sucede esto otro)

Si la primera condición (if) fue False, el programa puede revisar otras condiciones. elif significa "sino, si...". Puedes tener tantos elif como necesites.

- Ejemplo: elif esta_nublado: (Sino, si está nublado...) entonces voy al parque.

3. else (SINO... haz esto por defecto)

Si ninguna de las condiciones anteriores (if o elif) fue True, entonces el programa ejecuta el código que está en el else. Es el "plan B" o la acción por defecto.

- Ejemplo: else: (Sino, si nada de lo anterior se cumple...) entonces me quedo en casa.

Ejemplo Completo en la Vida Real:

Imagina que decides qué hacer el fin de semana según el clima:

SI (if) el día es soleado:

Voy a la playa.

SINO, SI (elif) el día está nublado:

Voy al parque.

SINO (else):

Me quedo en casa.



```
Untitled - TextEdit
File Edit View Help PYTHON

temperatura = 28

if temperatura > 25:
    print("¡Qué calor! Iré a la playa.")
elif temperatura > 15: # Si la temperatura no es
    mayor de 25, pero sí mayor de 15
    print("Clima agradable, ¡un paseo al
    parque!")
else: # Si ninguna de las anteriores fue True
    print("Hace frío, mejor me quedo en casa.")
```

Resultado: ¡Qué calor! Iré a la playa. (Porque 28 es mayor que 25).

En resumen, los condicionales if, elif, else le dan a tu programa la capacidad de pensar y reaccionar de forma diferente según lo que esté sucediendo, haciendo que sea mucho más inteligente y útil.



Bucles:
for
while (con break, continue, else)

Bucles: Repetir, repetir y repetir

Imagina que tienes que hacer algo muchas veces, como si fuera un juego. Los bucles son como una canción que te dice "¡hazlo otra vez!" o una regla que dice "¡sigue haciendo esto hasta que...!"

¡Claro que sí! Con gusto te explico los bucles como si fuera para un niño de 3 años, con ejemplos divertidos.

Bucles: Repetir, repetir y repetir

Imagina que tienes que hacer algo muchas veces, como si fuera un juego. Los bucles son como una canción que te dice "¡hazlo otra vez!" o una regla que dice "¡sigue haciendo esto hasta que...!"

1. for: "Hazlo por cada cosa"

Imagina que tienes una caja llena de juguetes. El bucle for es como si te dijeran:

"¡POR CADA juguete en la caja, dale un abrazo!"

LA HECHICERA
DEL CÓDIGO



PYTHON

La hechicera del código



```
1  for <v> jucle:
2      for [redacted] ;MIENTAS MIENTES esto verda!"
0);
2      while cuerda_gibraing();
3      while actua"= tomia_sibutar();
3      print(ior al telefone.
3);
6
2      while cuerda_gibraing(;
4      while actual = continue();
5      if dibbek.de-sigural = tomarins.esta_giirual()
);
while:
6      while cuerda_gibrobo = esta_pintar())
7          / !suena-el telefone.
);
8      primestar el telefone.

3      if ;Sáltte!! continue!;
6          / printa !break!

else:
1      "Si termiisat tolo! hague el poster..."
else:
1      "Si Termiatuna!, Huquia ssup! Hací vola postere."
);
```



LA HECHICERA
DEL CÓDIGO



PYTHON

La hechicera del código



```
juguetes = ["oso", "muñeca", "carro"]

for mi_juguete in juguetes:
    print(f"¡Abrazo a mi {mi_juguete}!")
```

Lo que hace:

Abrazo a mi oso!

Abrazo a mi muñeca!

Abrazo a mi carro!

2. while: "Sigue haciendo esto MIENTRAS sea verdad"

El bucle while es como cuando tu mamá te dice:

"MIENTRAS tu cuarto esté desordenado, ¡sigue recogiendo tus juguetes!"

Tú sigues recogiendo hasta que tu cuarto esté ordenado.



```
Untitled - TextEdit
File Edit View Help PYTHON

num_galletas = 3

while num_galletas > 0:
    print(f"¡Me como una galleta! Quedan {num_galletas} galletas.")
    num_galletas = num_galletas - 1 # Le quitamos 1 galleta
print("¡Se acabaron las galletas!")
```

Lo que hace:

¡Me como una galleta! Quedan 3 galletas.

¡Me como una galleta! Quedan 2 galletas.

¡Me como una galleta! Quedan 1 galletas.

¡Se acabaron las galletas!

Trucos especiales para while:

break (¡Para!): Imagina que estás comiendo galletas y de repente ¡te da dolor de panza! No quieres más. break es como decir "¡Para ya el bucle, no sigas!".



```
Untitled - TextEdit
File Edit View Help PYTHON

galletas = 5
while galletas > 0:
    print(f"Comiendo galleta... Quedan {galletas}.")
    if galletas == 3:
        print("¡Oh no, me duele la panza! ¡PARAR!")
        break # Aquí se detiene el bucle
    galletas -= 1
```

Comiendo galleta... Quedan 5.
Comiendo galleta... Quedan 4.
¡Oh no, me duele la panza! ¡PARAR!
continue (¡Sáltate esto!): Imagina que estás buscando tus juguetes, pero encuentras uno roto. Dices "¡Sáltate este, no lo recojo, busco el siguiente!". continue es como decir "¡Sáltate esta vez y ve a la siguiente!".



PYTHON

La hechicera del código



```
Untitled - TextEdit
File Edit View Help PYTHON

numeros = [1, 2, 3, 4, 5]
for num in numeros:
    if num == 3:
        print(f"¡El {num} es especial! ¡SALTAR!")
        continue # Se salta el resto del código para 3 y va al siguiente número
    print(f"Contando: {num}")
```

Contando: 1
Contando: 2
¡El 3 es especial! ¡SALTAR!
Contando: 4
Contando: 5
else (¡Cuando terminas!): Si el bucle while termina solito (porque la condición se hizo falsa), puedes decir "¡Ya que terminamos todo, hagamos otra cosa!".

```
Untitled - TextEdit
File Edit View Help PYTHON

contador = 0
while contador < 3:
    print(f"Contando... {contador}")
    contador += 1
else: # Esto se ejecuta SOLO si el bucle terminó normalmente
    print("¡Terminé de contar!")
```



Contando... 0
Contando... 1
Contando... 2
¡Terminé de contar!
¡Así funcionan los bucles!
Te ayudan a repetir cosas
sin tener que escribir lo
mismo una y otra vez.



PYTHON

La hechicera del código



Funciones útiles en bucles:

Función	Descripción	Ejemplo
range()	Crea una secuencia de números. Es como una lista que no existe en la memoria, pero que puedes usar para repetir un bucle un número específico de veces. Puedes decirle dónde empezar, dónde terminar y de cuánto en cuánto quieres que cuente.	for i in range(5): (Se repite 5 veces, con i tomando los valores 0, 1, 2, 3, 4)
enumerate()	Asocia un índice a cada elemento. Imagina que numeras cada artículo de una lista para no perder la cuenta. enumerate() te da la posición (el índice) y el valor de cada elemento.	for i, fruta in enumerate(['manzana', 'pera']): (i = 0, fruta = 'manzana'; i = 1, fruta = 'pera')
zip()	Combina dos o más listas elemento por elemento. Es como si unieras dos o más listas con un cierre, creando pares (o tuplas) de elementos que puedes recorrer juntos.	nombres = ['Ana', 'Juan'] edades = [25, 30] for n, e in zip(nombres, edades): (n = 'Ana', e = 25; n = 'Juan', e = 30)
min()	Encuentra el valor más bajo. Puedes usarla dentro de un bucle para comparar valores y saber cuál es el más pequeño.	numeros = [10, 5, 20] min(numeros) (Devuelve 5)
max()	Encuentra el valor más alto. Es el opuesto de min().	numeros = [10, 5, 20] max(numeros) (Devuelve 20)
sum()	Suma todos los elementos.	numeros = [10, 5, 20] sum(numeros) (Devuelve 35)



PYTHON

La hechicera del código



List comprehension

¿Qué es List Comprehension?

Imagina que tienes que hacer una lista nueva a partir de una lista que ya tienes. Tradicionalmente, lo harías con un bucle for, como si estuvieras llenando una canasta uno por uno.

List Comprehension es como una máquina mágica que hace lo mismo, pero en una sola línea de código, más rápido y de forma más elegante. Es una manera "inteligente" de crear listas.

Cómo funciona (La receta mágica)

La estructura es muy sencilla: [expresión for elemento in lista]

Parte	Significado	Ejemplo
Expresión	Lo que quieres hacer con cada elemento.	$i * 2$ (multiplicar por 2)
for	La palabra mágica que inicia el bucle.	for
elemento	Una variable temporal para cada elemento de la lista.	i
in lista	La lista original que quieres recorrer.	números Exportar a Hojas de cálculo



Claro, te explico el concepto de List Comprehension de manera muy simple.

¿Qué es List Comprehension?

Imagina que tienes que hacer una lista nueva a partir de una lista que ya tienes. Tradicionalmente, lo harías con un bucle for, como si estuvieras llenando una canasta uno por uno.

List Comprehension es como una máquina mágica que hace lo mismo, pero en una sola línea de código, más rápido y de forma más elegante. Es una manera "inteligente" de crear listas.

Cómo funciona (La receta mágica)

La estructura es muy sencilla: [expresión for elemento in lista]

Parte Significado Ejemplo

Expresión Lo que quieres hacer con cada elemento. $i * 2$ (multiplicar por 2)

for La palabra mágica que inicia el bucle. for

elemento Una variable temporal para cada elemento de la lista. i

in lista La lista original que quieres recorrer.
numeros

Exportar a Hojas de cálculo

Ejemplo: Sin List Comprehension vs. Con List Comprehension

Supongamos que tienes una lista de números y quieres crear una nueva lista con cada número multiplicado por 2.

Método 1: Usando un bucle for (La forma tradicional)



PYTHON

La hechicera del código



```
Untitled - TextEdit
File Edit View Help PYTHON

numeros = [1, 2, 3, 4, 5]
numeros_doble = [] # Lista vacía para
guardar los resultados

for numero in numeros:
    numeros_doble.append(numero * 2)

print(numeros_doble)
# Salida: [2, 4, 6, 8, 10]
```

Método 2: Usando List Comprehension (La forma mágica)

```
Untitled - TextEdit
File Edit View Help PYTHON

numeros = [1, 2, 3, 4, 5]

numeros_doble = [numero * 2 for numero in
numeros]

print(numeros_doble)
# Salida: [2, 4, 6, 8, 10]
```

Como puedes ver, List Comprehension logra el mismo resultado de una manera mucho más limpia y compacta. Es una herramienta muy poderosa y común en Python.



5. Funciones

Funciones: El arte de organizar tu código

Claro, con gusto te ofrezco un resumen de estos conceptos clave en Python.

Funciones: El arte de organizar tu código

Las funciones son como recetas que agrupan un conjunto de instrucciones. En lugar de escribir el mismo código una y otra vez, lo pones en una función y la llamas cuando la necesitas.

Concepto	Descripción
Crear funciones (def)	Usas la palabra def para definir tu "receta" y le das un nombre. Todo lo que esté dentro de la función (con sangría) es parte de ella.
return y funciones dinámicas	Con return, tu función puede entregar un resultado. Es como si una calculadora te "devuelve" el número que calculó. Esto las hace dinámicas, ya que pueden generar un resultado diferente cada vez que las llamas con distintos datos.
Interacción entre funciones	Una función puede llamar a otra función. Es como si una receta principal te dijera: "Ahora, sigue la receta del pastel, y cuando termines, trae el resultado para continuar".



Argumentos y flexibilidad

Los argumentos son los ingredientes que le das a una función para que trabaje.

Concepto	Descripción
*args (Argumentos indefinidos)	Te permite pasar un número variable de argumentos a una función. Python los agrupa en una tupla. Es como si una receta dijera: "Necesito especias, ¡puedes darme tantas como quieras!".
**kwargs (Argumentos de palabra clave)	Similar a *args, pero para argumentos que tienen un nombre (clave=valor). Python los agrupa en un diccionario. Es como si la receta dijera: "Dame los ingredientes con su nombre: sal=una_pizca, azucar=una_cucharada".

Herramientas clave

Concepto	Descripción
Funciones anónimas (lambda)	Son funciones muy cortas que puedes definir en una sola línea, sin darles un nombre formal. Son útiles para tareas rápidas y simples.
Documentación (help() y __doc__)	Python te permite documentar tu código. <code>__doc__</code> es donde guardas la descripción de la función, y <code>help()</code> es la herramienta que te ayuda a leer esa documentación. Es como un manual de usuario para tu código.



6. Manejo de Archivos y Directorios

📁 Manejo de Archivos y Directorios

Estos conceptos te permiten que tus programas interactúen con archivos y carpetas en tu computadora, como si fueran un bibliotecario y un archivador.

Abrir y Leer Archivos (`open()`, `read()`, `write()`, `with`):

- `open()` es la función que te permite abrir un archivo. Le das el nombre y si quieres leerlo ('r') o escribir en él ('w').
- `read()` es para leer el contenido del archivo.
- `write()` es para escribir nuevo contenido.
- `with`: Es la forma recomendada de trabajar con archivos. Se asegura de que el archivo se cierre automáticamente, incluso si hay errores. Es como si el bibliotecario siempre pusiera el libro en su lugar cuando terminas de usarlo.

Manipular Directorios (`os`, `pathlib`):

- Son módulos (bibliotecas) de Python que te dan herramientas para trabajar con carpetas y rutas de archivos.
- `os` es el módulo tradicional y te permite crear, borrar y mover directorios.
- `pathlib` es el módulo moderno, más sencillo e intuitivo, que representa las rutas como objetos para manipularlas de forma más segura.

Ejemplo de Arquitectura de Directorios:

- La arquitectura que mencionas (`C:\Users\Isela\PythonProjects\`) es una forma de organizar tu proyecto de manera ordenada, separando el código principal (`main.py`), los datos (`data/`), las imágenes (`images/`) y los módulos de ayuda (`modules/`). Esto es una buena práctica para proyectos grandes.

Limpiar Consola (`os.system('cls' or 'clear')`):

- Esta función ejecuta un comando del sistema operativo.
- En Windows, `os.system('cls')` borra la pantalla.
- En sistemas como Linux o macOS, usas `os.system('clear')`.
- Es útil para mantener la salida del programa ordenada y fácil de leer.



PYTHON

La hechicera del código



```
Untitled - TextEdit
File Edit View Help PYTHON
C:\Users\Isela\PythonProjects\
|-- main.py
|-- data\
|   |-- dataset.csv
|   |-- notes.txt
|-- images\
|   |-- logo.png
|   |-- photo.jpg
|-- modules\
    |-- helper.py
```

LA HECHICERA
DEL CÓDIGO



7. Programación Orientada a Objetos (POO)

Programación Orientada a Objetos (POO)

La POO es una forma de programar que organiza el código alrededor de "objetos" en lugar de solo funciones y lógica. Imagina un objeto como cualquier cosa del mundo real, por ejemplo, un perro 🐕.

Clases y objetos

Clase: Es el "molde" o el "plano" para crear objetos. Describe qué atributos y métodos tendrán. Por ejemplo, la clase Perro define que todos los perros tienen un color, un nombre y que pueden ladrar.

Objeto: Es una "instancia" o una "copia" específica de la clase. Es el perro real que creaste a partir del molde. Un objeto podría ser mi_perro con el nombre "Fido" y color "marrón".

Atributos y métodos

Atributos: Son las características o propiedades de un objeto. Son los datos que lo describen. Para nuestro perro, los atributos serían nombre, color, raza.

Métodos: Son las acciones o comportamientos que un objeto puede realizar. Para nuestro perro, los métodos serían ladrar(), correr(), comer().



Tipos de métodos

Métodos de instancia: Pertenecen a un objeto específico y necesitan que el objeto exista para ser usados.

Métodos de clase: Pertenecen a la clase en sí, no a una instancia específica. Afectan a la clase completa.

Métodos estáticos: Son métodos que no necesitan ni una instancia ni la clase para funcionar. Son funciones "normales" que se agrupan dentro de una clase por conveniencia.

Herencia y herencia extendida

Herencia: Permite que una clase "hija" herede los atributos y métodos de una clase "padre". Por ejemplo, una clase Gato y una clase Perro pueden heredar de una clase Animal que tiene los atributos patas y nombre, y el método comer().

Herencia extendida: La clase "hija" no solo hereda, sino que también puede añadir nuevos atributos o métodos, o modificar los que heredó.

Polimorfismo

Es la capacidad de los objetos de diferentes clases para responder de manera diferente al mismo método. Por ejemplo, si tienes un método **hacer_sonido()**, el perro respondería con un "Guau" y el gato con un "Miau", aunque ambos llamen al mismo método.

Métodos especiales

Son métodos que se llaman automáticamente en ciertas situaciones y sus nombres comienzan y terminan con dos guiones bajos (`__`).

`__init__`: El "constructor". Se ejecuta cuando creas un objeto. Se usa para inicializar sus atributos.

`__str__`: Devuelve una representación del objeto en forma de cadena de texto, ideal para imprimirlo y que sea legible para el usuario.

`__len__`: Devuelve la longitud del objeto cuando usas la función `len()`.



8. Paquetes, Módulos y Librerías

Paquetes, Módulos y Librerías

Estos son conceptos clave para organizar y reutilizar código en Python. Piensa en ellos como piezas de un set de construcción que puedes usar en tus proyectos.

Módulo: Un simple archivo de Python (.py) que contiene funciones, clases y variables. Es la unidad más básica.

Paquete: Una colección de módulos organizados en directorios (carpetas). Un paquete te ayuda a estructurar proyectos grandes.

Librería: Un término más amplio que se usa para referirse a una colección de módulos y paquetes que hacen una tarea específica.

Uso práctico:

Instalar paquetes (pip install): pip es la herramienta estándar para descargar e instalar paquetes de otras personas. Por ejemplo, pip install numpy te permite usar la librería numpy.

Crear y usar módulos propios: Si tienes un archivo utilidades.py con una función, puedes usarlo en tu programa principal con la línea import utilidades.

Importar librerías estándar: Python ya trae muchas librerías listas para usar, como math para matemáticas o random para números aleatorios. Solo necesitas importarlas: import random.



Módulos avanzados

Estos son algunos de los módulos más útiles que vienen con Python:

collections: Herramientas para tipos de datos especializados. Por ejemplo, Counter que cuenta la frecuencia de elementos en una lista, o defaultdict que te ayuda a trabajar con diccionarios de forma más sencilla.

os y shutil: Módulos para interactuar con el sistema operativo. os es para manejar directorios y archivos, mientras que shutil es para operaciones de alto nivel, como copiar o mover archivos y carpetas de forma segura.

re (expresiones regulares): Un módulo muy poderoso para buscar, manipular y validar patrones en cadenas de texto. Es útil para, por ejemplo, validar un correo electrónico o un número de teléfono.

zipfile: Te permite comprimir y descomprimir archivos en formato ZIP. Es muy útil para manejar grandes volúmenes de datos.



9. Testing y Depuración

Esta es la fase en la que revisas y arreglas tu código para que funcione correctamente y sin errores. Es como un detective que busca pistas para encontrar fallas.

Concepto	Descripción
Manejo de errores (try, except, finally)	Es una forma de "probar" tu código de manera segura. • try: Pones aquí el código que podría fallar. • except: Si algo falla en try, el programa va a except para ejecutar un código alternativo (por ejemplo, mostrar un mensaje de error legible). • finally: Este código siempre se ejecuta, sin importar si hubo un error o no. Es útil para cerrar archivos o limpiar recursos.
Buscar errores con pylint	Es una herramienta que revisa tu código y te da sugerencias para mejorarlo. No solo encuentra errores, sino que también te ayuda a seguir un estilo de codificación limpio y profesional, haciendo que sea más fácil de leer para otras personas.
Pruebas unitarias con unittest	Las pruebas unitarias son pequeñas pruebas automatizadas que verifican si una parte específica de tu código (una "unidad", como una función) funciona como esperas. unittest es una librería estándar de Python que te ayuda a escribir y ejecutar estas pruebas de forma organizada.



10. Temas Avanzados

Estos conceptos te llevan más allá de lo básico, permitiéndote escribir código más potente, flexible y optimizado.

Concepto	Descripción
Decoradores	Son funciones que "envuelven" a otras funciones para añadirles una funcionalidad extra sin modificar su código original. Por ejemplo, puedes crear un decorador para medir cuánto tiempo tarda en ejecutarse una función.
Generadores (yield)	Son como funciones que pausan su ejecución y "devuelven" un valor temporalmente. Usan la palabra clave yield. Son muy eficientes para trabajar con grandes cantidades de datos, ya que no guardan toda la información en memoria al mismo tiempo.
Medir el tiempo de ejecución (time, timeit)	Estos módulos te permiten saber qué tan rápido (o lento) es tu código. time es útil para medir tiempos de forma simple, mientras que timeit es más preciso y está diseñado para probar fragmentos de código.
Tkinter - GUI básica	Es la librería estándar de Python para crear interfaces gráficas de usuario (GUI). Te permite diseñar ventanas, botones, campos de texto y otros elementos para que tu programa sea más interactivo para el usuario.
Extraer elementos de una clase (_dict_, getattr)	Son herramientas para inspeccionar objetos. _dict_ es un diccionario que contiene todos los atributos de un objeto, y getattr() es una función que te permite acceder a los atributos de un objeto usando su nombre como una cadena de texto.
Extraer imágenes con librerías externas	Para tareas más complejas como extraer imágenes de un PDF, manipular fotos o generar gráficos, Python se apoya en librerías de terceros (que se instalan con pip), como Pillow. Estas bibliotecas extienden las capacidades del lenguaje de forma poderosa.



11. Manejo de entornos virtuales (venv)

Imagina que estás trabajando en diferentes proyectos, y cada uno necesita un conjunto de herramientas distinto. Por ejemplo:

El Proyecto A necesita la versión 1 de una herramienta llamada "Pandas".

El Proyecto B necesita la versión 2 de esa misma herramienta.

Si instalas las herramientas en tu computadora de forma global, podrías causar un conflicto, ya que una versión podría "pisar" a la otra.

Un entorno virtual es como crear una caja de herramientas exclusiva y aislada para cada proyecto. Es un espacio privado donde puedes instalar las librerías que necesitas sin que afecten a tus otros proyectos o a la instalación global de Python en tu computadora.

LA HECHICERA
DEL CÓDIGO



PYTHON

La hechicera del código



	Sin Entorno Virtual	Con Entorno Virtual
Concepto	Todas las herramientas están en un solo lugar (tu "caja de herramientas principal").	Cada proyecto tiene su propia caja de herramientas privada.
Riesgo	Los proyectos pueden interferir entre sí.	Los proyectos están aislados y no se afectan.
Organización	Difícil de gestionar las dependencias de cada proyecto.	Fácil de replicar y compartir un proyecto con todas sus dependencias. Exportar a Hojas de cálculo

Comando	Descripción
<code>python -m venv nombre_del_entorno</code>	Crea una nueva "caja de herramientas" (un entorno virtual).
<code>source nombre_del_entorno/bin/activate</code>	En Mac/Linux: Activa el entorno virtual para que puedas usarlo.
<code>nombre_del_entorno\Scripts\activate</code>	En Windows: Activa el entorno virtual.
<code>pip install nombre_de_libreria</code>	Instala una librería, pero solo dentro de la caja de herramientas de tu proyecto.
<code>deactivate</code>	Te saca del entorno virtual y te regresa a tu instalación global de Python.



12. Buenas prácticas y estilo de código (PEP8)

Las buenas prácticas y el estilo de código (PEP8) son un conjunto de reglas y recomendaciones para escribir código en Python que sea más legible, limpio y consistente. Es como la guía de estilo para escribir de forma profesional y ordenada.

¿Por qué son importantes?

Legibilidad: Un código limpio es mucho más fácil de leer, entender y mantener, tanto para ti como para otros programadores.

Consistencia: Si todos siguen las mismas reglas, el código se ve uniforme, sin importar quién lo haya escrito.

Colaboración: Facilita el trabajo en equipo, ya que no tienes que descifrar un estilo de codificación diferente en cada proyecto.

Principales recomendaciones de la PEP8



PYTHON

La hechicera del código



Regla	Descripción	Ejemplo de la vida real
GUI	Significa Graphical User Interface. Es lo que te permite interactuar con un programa usando elementos visuales como ventanas, iconos, menús, etc., en lugar de solo escribir comandos.	La pantalla de tu celular o la ventana de un programa en tu computadora.
Tkinter	Es la librería estándar de Python para construir esas GUI. Piensa en ella como una caja de herramientas con todas las piezas (ventanas, botones, etiquetas) que necesitas.	Un set de LEGO para construir casas pequeñas.
Ventana (root)	Es el contenedor principal de tu programa. Como el marco de una fotografía donde irán todos los demás elementos.	La pantalla completa de una aplicación.
Widgets	Son los "elementos" que pones dentro de la ventana. Cada botón, cada etiqueta, cada caja de texto es un widget.	Los botones, perillas o pantallas de un radio.
Eventos	Son las acciones que el usuario realiza (hacer clic en un botón, escribir en una caja). Tkinter te permite que tu programa "reaccione" a estos eventos.	Cuando presionas un botón en tu celular y algo sucede.



PYTHON

La hechicera del código



13.Tkinter (GUI)

Tkinter: Construyendo ventanas y botones para tus programas

Imagina que quieres que tu programa de Python no sea solo texto en una pantalla negra, sino que tenga una ventana bonita con botones que puedas presionar, cajas donde escribir texto y etiquetas que muestren información.

Tkinter es el "kit de construcción" que Python ya trae incorporado para hacer precisamente eso: crear Interfaces Gráficas de Usuario (GUI).

LA HECHICERA
DEL CÓDIGO



PYTHON

La hechicera del código



Regla	Descripción
Sangría	Usa 4 espacios por cada nivel de sangría. Nunca uses tabulaciones.
Longitud de línea	Limita cada línea a un máximo de 79 caracteres. Esto hace que el código sea fácil de leer en cualquier pantalla.
Nombres de variables	Usa nombres descriptivos en minúsculas, separados por guiones bajos (_). Por ejemplo: nombre_de_usuario.
Nombres de funciones	Sigue las mismas reglas que para las variables. Ejemplo: calcular_total().
Nombres de clases	Usa el formato "CamelCase" (cada palabra con mayúscula inicial). Ejemplo: MiClaseDeEjemplo.
Espacios en blanco	Añade espacios alrededor de los operadores (=, +, -) y después de las comas.
Comentarios	Mantén los comentarios claros y concisos, y úsalos para explicar por qué el código hace algo, no qué hace.
Líneas en blanco	Usa líneas en blanco para separar funciones y bloques lógicos de código.



¿Cómo se ve en Python (ejemplo muy simple)?

```
import tkinter as tk # Importamos el "kit de construcción"

ventana = tk.Tk() # Creamos nuestra primera ventana (el marco)
ventana.title("Mi Primera App") # Le ponemos un título a la ventana

etiqueta = tk.Label(ventana, text="¡Hola, mundo!") # Creamos una
etiqueta.de_texto
etiqueta.pack() # Colocamos la etiqueta dentro de la ventana

boton = tk.Button(ventana, text="Haz clic") # Creamos un botón
boton.pack() # Colocamos el botón

ventana.mainloop() # Hacemos que la ventana se quede abierta y
esperando que el usuario haga algo
```

Cuando ejecutes este código, ¡aparecerá una pequeña ventana con el texto "¡Hola, mundo!" y un botón!

En resumen, Tkinter te permite darle una "cara" visual a tus programas de Python, haciéndolos mucho más amigables e interactivos para cualquier persona.



14.Extraer imágenes (Pillow)

Pillow: La forma fácil de trabajar con imágenes en Python

Pillow es una de las librerías más populares y poderosas en Python para abrir, manipular y guardar imágenes en diferentes formatos. Piensa en ella como un editor de imágenes súper potente, pero en forma de código.

Paso a paso para extraer una imagen

El proceso general para trabajar con imágenes usando Pillow es muy sencillo:

1. Instalación

Primero, necesitas instalar la librería. Como no viene incluida con Python, debes usar pip:

```
pip install Pillow
```

2. Cargar la imagen

Para abrir un archivo de imagen, usas la función `Image.open()`.



```
Untitled - TextEdit
File Edit View Help PYTHON

from PIL import Image

# Reemplaza 'nombre_de_mi_imagen.jpg'
# con la ruta de tu archivo
imagen =
Image.open('nombre_de_mi_imagen.jpg')
```

3. Manipular o extraer (opcional)

Una vez que la imagen está cargada en una variable, puedes hacer muchas cosas con ella. Por ejemplo, para extraer una parte de la imagen, usas el método `crop()`.

```
Untitled - TextEdit
File Edit View Help PYTHON

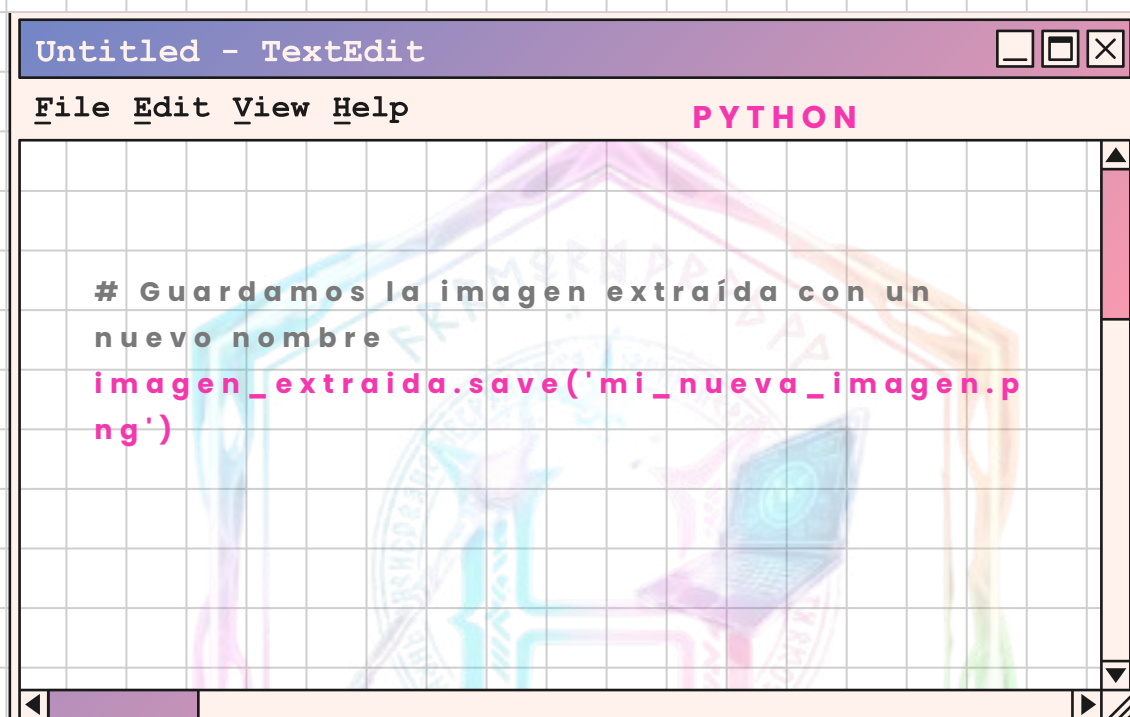
# Definimos las coordenadas para el
# recorte (izquierda, arriba, derecha, abajo)
caja_de_recorte = (100, 100, 400, 400)

# Extraemos la parte de la imagen dentro
# de esa caja
imagen_extraida =
imagen.crop(caja_de_recorte)
```



4. Guardar la imagen

Finalmente, si quieres guardar la imagen modificada o extraída, usas el método `save()`.



```
Untitled - TextEdit
File Edit View Help PYTHON

# Guardamos la imagen extraída con un
nuevo nombre
imagen_extraida.save('mi_nueva_imagen.p
ng')
```

Resumen:

Paso 1: Instala Pillow.

Paso 2: Abre la imagen con `Image.open()`.

Paso 3: Usa `crop()` para extraer la parte que te interesa.

Paso 4: Guarda el resultado con `save()`.

Pillow es la herramienta perfecta para cualquier tarea relacionada con imágenes que necesites hacer con Python, desde extraer partes hasta redimensionar, rotar o aplicar filtros.



15.Pandas Manipulación de datos

Pandas: El Excel de Python para analizar datos 🇮🇹
Imagina que tienes una gran cantidad de datos, como una tabla de Excel con filas y columnas. Pandas es la librería de Python que te permite trabajar con esos datos de forma muy eficiente, como si fuera un superpoderoso Excel programable.

Estructuras de datos clave de Pandas

Pandas se basa en dos estructuras de datos principales:

1. Series

Una Series es como una sola columna de una tabla de Excel. Es una lista de datos con un índice.

Ejemplo: Una lista de las edades de varias personas.

```
0    25
1    30
2    22
dtype: int64
```

2. DataFrame

Un DataFrame es como una tabla completa de Excel, con filas y columnas. Es la estructura más usada en Pandas y es ideal para datos tabulares.

Ejemplo: Una tabla con nombres, edades y ciudades de personas.



PYTHON

La hechicera del código



	Nombre	Edad	Ciudad
0	Simba	1	Simbalandia
1	Luffy	13	One Pice
2	Inaya	128	Nevermore

¿Para qué se usa Pandas?

Pandas es la herramienta fundamental para el análisis de datos. Te permite:

Leer datos: Puede leer archivos de muchos formatos (CSV, Excel, JSON, etc.) y convertirlos en un DataFrame.

Limpiar datos: Te ayuda a encontrar y manejar datos faltantes, duplicados o incorrectos.

Filtrar y seleccionar: Puedes buscar y seleccionar filas o columnas específicas con gran facilidad.

Agrupar y resumir: Te permite hacer cálculos como promedios, sumas o contar elementos, agrupando la información por categorías.

En resumen, si trabajas con datos en forma de tablas, Pandas es la herramienta esencial que te ahorrará muchísimo tiempo y te permitirá hacer análisis complejos de manera programática.



PYTHON

La hechicera del código



Ejemplo de uso de Pandas: Un listado de productos

Imagina que tienes una lista de productos en tu tienda y quieres analizarlos. En lugar de usar una hoja de cálculo, lo harás con código Python usando Pandas.

Paso 1: Importar la librería Primero, necesitas importar Pandas, que por convención se usa con el alias pd.

```
import pandas as pd
```

Paso 2: Crear el DataFrame (tu tabla de datos)
Ahora, creamos un diccionario con los datos. Cada clave será una columna, y los valores serán los datos de esa columna. Luego, usamos `pd.DataFrame()` para convertirlo en una tabla de Pandas.



PYTHON

La hechicera del código



```
Untitled - TextEdit
File Edit View Help PYTHON

datos = {
    'Producto': ['Laptop', 'Mouse', 'Teclado',
                'Monitor', 'Audífonos'],
    'Precio': [1200, 25, 75, 300, 50],
    'Stock': [15, 50, 30, 10, 25]
}

df = pd.DataFrame(datos)

print("Tabla de datos (DataFrame):")
print(df)
```

	Producto	Precio	Stock
0	Laptop	1200	15
1	Mouse	25	50
2	Teclado	75	30
3	Monitor	300	10
4	Audífonos	50	25



PYTHON

La hechicera del código



Paso 3: Realizar un análisis simple

Ahora que tienes los datos en un DataFrame, puedes hacer preguntas y obtener respuestas fácilmente.

Seleccionar una columna:

```
Untitled - TextEdit
File Edit View Help PYTHON

# Muestra solo la columna
'Precio'
print("\nSolo la columna de
precios:")
print(df['Precio'])
```

Filtrar datos:

```
Untitled - TextEdit
File Edit View Help PYTHON

# Muestra los productos con un
precio menor a $100
print("\nProductos con precio
menor a $100:")
productos_economicos =
df[df['Precio'] < 100]
print(productos_economicos)
```



Calcular una estadística:

```
Untitled - TextEdit
File Edit View Help PYTHON

# Calcula el precio promedio de
# todos los productos
precio_promedio =
df['Precio'].mean()
print(f"\nEl precio promedio de los
productos es: ${precio_promedio}")
```

Como puedes ver, Pandas te permite manipular los datos de manera muy intuitiva y poderosa, similar a como lo harías en Excel, pero con la flexibilidad y el poder de la programación.



16.Scikit-learn para algoritmos de ML

Scikit-learn: La navaja suiza del Machine Learning 🧙‍♂️

Imagina que quieres enseñarle a una computadora a reconocer patrones y tomar decisiones, como predecir el precio de una casa o clasificar imágenes. Ese proceso se llama aprendizaje automático (ML).

Scikit-learn (a menudo abreviado como sklearn) es la librería más popular en Python para hacer precisamente eso. No es una librería para crear redes neuronales complejas, sino que se especializa en los algoritmos de ML más comunes y útiles, lo que la hace perfecta para principiantes y expertos por igual.

¿Qué puedes hacer con Scikit-learn?

Puedes usarla para resolver los problemas de ML más comunes:

- **Clasificación:** Agrupar datos en categorías.

Por ejemplo, clasificar correos electrónicos como "spam" o "no-spam".

- **Regresión:** Predecir un valor numérico continuo. Por ejemplo, predecir el precio de una casa basándose en su tamaño y ubicación.

- **Clustering:** Encontrar grupos ocultos en los datos. Por ejemplo, agrupar a clientes con comportamientos de compra similares para segmentar el mercado.

Reducción de dimensionalidad: Simplificar conjuntos de datos muy grandes.

¿Cómo se usa?

Scikit-learn tiene una estructura muy consistente y fácil de usar. El proceso general para la mayoría de los modelos es el siguiente:

- **Importar:** Importas la clase del modelo que quieres usar.

Instanciar: Creas una "instancia" del modelo (el objeto).

- **Entrenar:** Usas el método `.fit()` para "entrenar" el modelo con tus datos.

- **Predecir:** Usas el método `.predict()` para hacer predicciones con nuevos datos.

Ejemplo Sencillo de Clasificación

Supongamos que quieres clasificar si una flor es una especie u otra basándote en la longitud de sus pétalos.



PYTHON

La hechicera del código



Untitled - TextEdit

File Edit View Help

PYTHON

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets

# 1. Cargar los datos de ejemplo
iris = datasets.load_iris()
X, y = iris.data, iris.target

# 2. Separar datos para entrenamiento y prueba
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba =
train_test_split(X, y, test_size=0.3)
```

Untitled - TextEdit

File Edit View Help

PYTHON

```
# 3. Crear el modelo de clasificación (K-vecinos más
ceranos)
modelo = KNeighborsClassifier()

# 4. Entrenar el modelo con los datos de entrenamiento
modelo.fit(X_entrenamiento, y_entrenamiento)

# 5. Hacer una predicción con un nuevo dato
nueva_flor = [[5.0, 3.5, 1.3, 0.3]]
prediccion = modelo.predict(nueva_flor)

print(f"La predicción es:
{iris.target_names[prediccion[0]]}")
```

Como puedes ver, Scikit-learn hace que sea muy fácil implementar y experimentar con algoritmos de Machine Learning, simplificando un proceso que de otra forma sería muy complejo.



17. NumPy para cálculos numéricos

NumPy: El poder de los números para Python 🚀

Imagina que tienes que hacer operaciones matemáticas muy complejas con listas de números, como si tuvieras una tabla con miles de datos y quisieras sumarlos, restarlos o multiplicarlos rápidamente.

Python por sí solo puede hacerlo, pero es lento. Ahí es donde entra NumPy (Numerical Python).

NumPy es una librería fundamental que le da a Python un superpoder para trabajar con números. Lo hace a través de una estructura de datos llamada ndarray (N-dimensional array), que es como una lista, pero mucho más rápida y eficiente para operaciones numéricas.

¿Qué hace a NumPy tan especial?

Velocidad: Los ndarrays de NumPy son mucho más rápidos que las listas de Python para operaciones matemáticas.

Operaciones Vectorizadas: Te permite aplicar una operación a un arreglo completo de una sola vez, sin necesidad de usar bucles for. Por ejemplo, puedes sumar dos arreglos completos con un solo signo +.

Funciones matemáticas: Viene con un vasto conjunto de funciones matemáticas listas para usar, como seno, coseno, logaritmos, etc., que se aplican a todo el arreglo.

Ejemplo Sencillo

Imagina que quieres duplicar todos los números en una lista.

Sin NumPy (Usando una lista normal de Python)



PYTHON

La hechicera del código



```
Untitled - TextEdit
File Edit View Help
PYTHON

lista = [1, 2, 3, 4, 5]
lista_doble = []
for numero in lista:
    lista_doble.append(numero * 2)

print(lista_doble)
```

Salida: [2, 4, 6, 8, 10]
Con NumPy (Usando un ndarray)

```
Untitled - TextEdit
File Edit View Help
PYTHON

import numpy as np # Por convención se usa np

# Convertimos la lista a un ndarray de NumPy
arreglo = np.array([1, 2, 3, 4, 5])

# Realizamos la operación de forma vectorizada
(muy rápida)
arreglo_doble = arreglo * 2

print(arreglo_doble)
```

Salida: [2 4 6 8 10]

Como puedes ver, el código es más corto, claro y, lo más importante, ¡muchísimo más rápido cuando trabajas con millones de datos! Por eso, NumPy es una de las librerías más importantes para la ciencia de datos y el machine learning en Python.



18. Matplotlib y Seaborn para visualización de datos

Matplotlib y Seaborn: Pintando tus datos para entenderlos mejor 📊🎨

Imagina que tienes muchísimos datos (números, palabras, etc.), y verlos así nomás no te dice nada. Para realmente entender lo que está pasando, necesitas "dibujarlos".

Matplotlib y Seaborn son dos librerías de Python que te permiten hacer exactamente eso: convertir tus datos en gráficos, diagramas y visualizaciones bonitas y fáciles de entender.

1. Matplotlib: El artista fundamental

¿Qué es? Es la librería base para crear gráficos en Python. Piensa en ella como tu lienzo, tus pinceles y tus pinturas. Te da control total para dibujar casi cualquier tipo de gráfico, desde los más simples hasta los más complejos.

¿Para qué sirve?

Dibujar líneas (para ver tendencias).

Hacer barras (para comparar cantidades).

Crear puntos dispersos (para ver relaciones entre dos cosas).

Hacer histogramas (para ver cómo se distribuyen tus datos).

Características: Requiere un poco más de código para lograr un resultado estético, pero te da muchísima flexibilidad.

Ejemplo de Matplotlib:



PYTHON

La hechicera del código



Untitled - TextEdit

File Edit View Help

PYTHON

```
import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5]
```

```
y = [2, 4, 1, 3, 5]
```

```
plt.plot(x, y) # Dibujamos una línea
```

```
plt.xlabel("Eje X") # Ponemos nombre al eje X
```

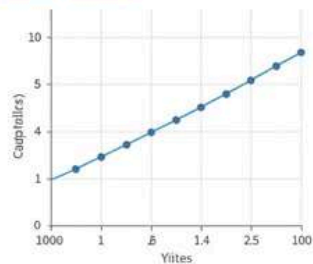
```
plt.ylabel("Eje Y") # Ponemos nombre al eje Y
```

```
plt.title("Gráfico de Líneas Simple") # Título del gráfico
```

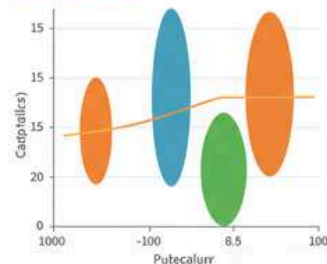
```
plt.show() # Mostramos el gráfico
```

Matplotlib and Seaborn

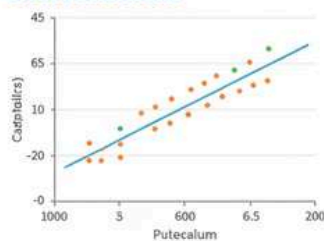
1. Matplotlib:



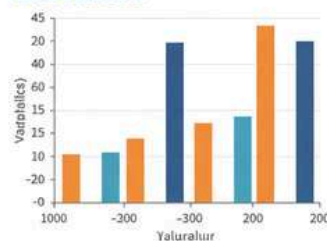
2. Inas dato



Charactisiticas



2. Seaborn





2. Seaborn: El decorador de gráficos

¿Qué es? Seaborn está construido sobre Matplotlib. Piensa en él como un "decorador de interiores" para tus gráficos. Te ayuda a hacer gráficos más bonitos, complejos y estadísticos con menos código.

¿Para qué sirve?

Hacer gráficos de cajas, violines (para ver distribuciones).

Mapas de calor (para ver correlaciones).

Gráficos de dispersión con líneas de regresión.

Características: Es excelente para explorar relaciones entre variables y produce gráficos estéticamente agradables por defecto. Funciona muy bien con DataFrames de Pandas.

Ejemplo de Seaborn:

Untitled - TextEdit

File Edit View Help

PYTHON

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

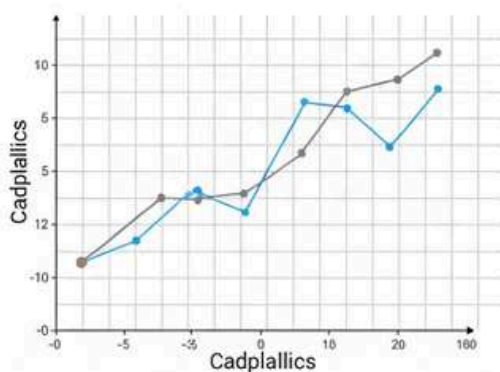
# Datos de ejemplo
datos = {'Edad': [20, 22, 25, 30, 35, 28],
         'Salario': [30000, 35000, 40000, 50000, 60000, 45000]}
df = pd.DataFrame(datos)

# Dibujamos un gráfico de dispersión con Seaborn
sns.scatterplot(x='Edad', y='Salario', data=df)
plt.title("Relación Edad y Salario")
plt.show()
```

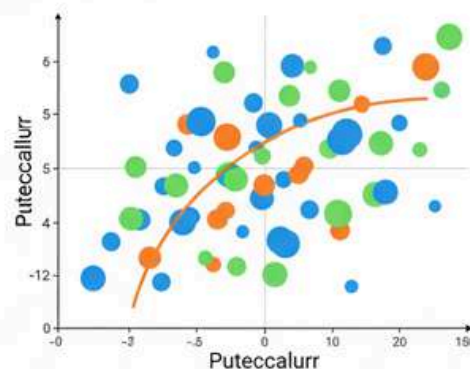


Matplotlib and Seaborn

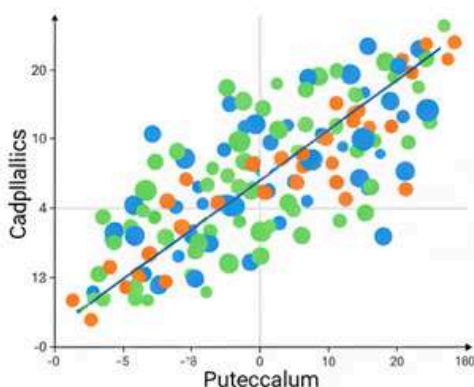
Matplotlib



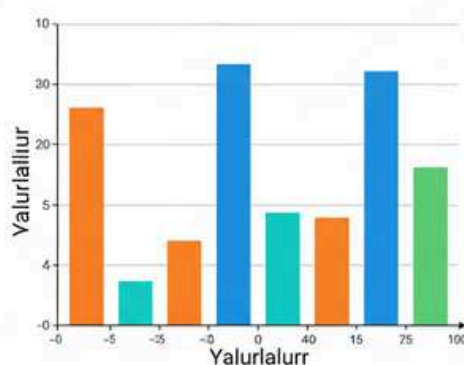
Inas dato



Charactisisticas



Seaborn



En resumen:

Matplotlib te da las herramientas básicas para construir cualquier gráfico.

Seaborn te facilita crear gráficos estadísticos complejos y estéticamente atractivos con menos esfuerzo, ideal para el análisis de datos.

Ambas librerías trabajan muy bien juntas y son esenciales para cualquier persona que trabaje con datos en Python.



PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

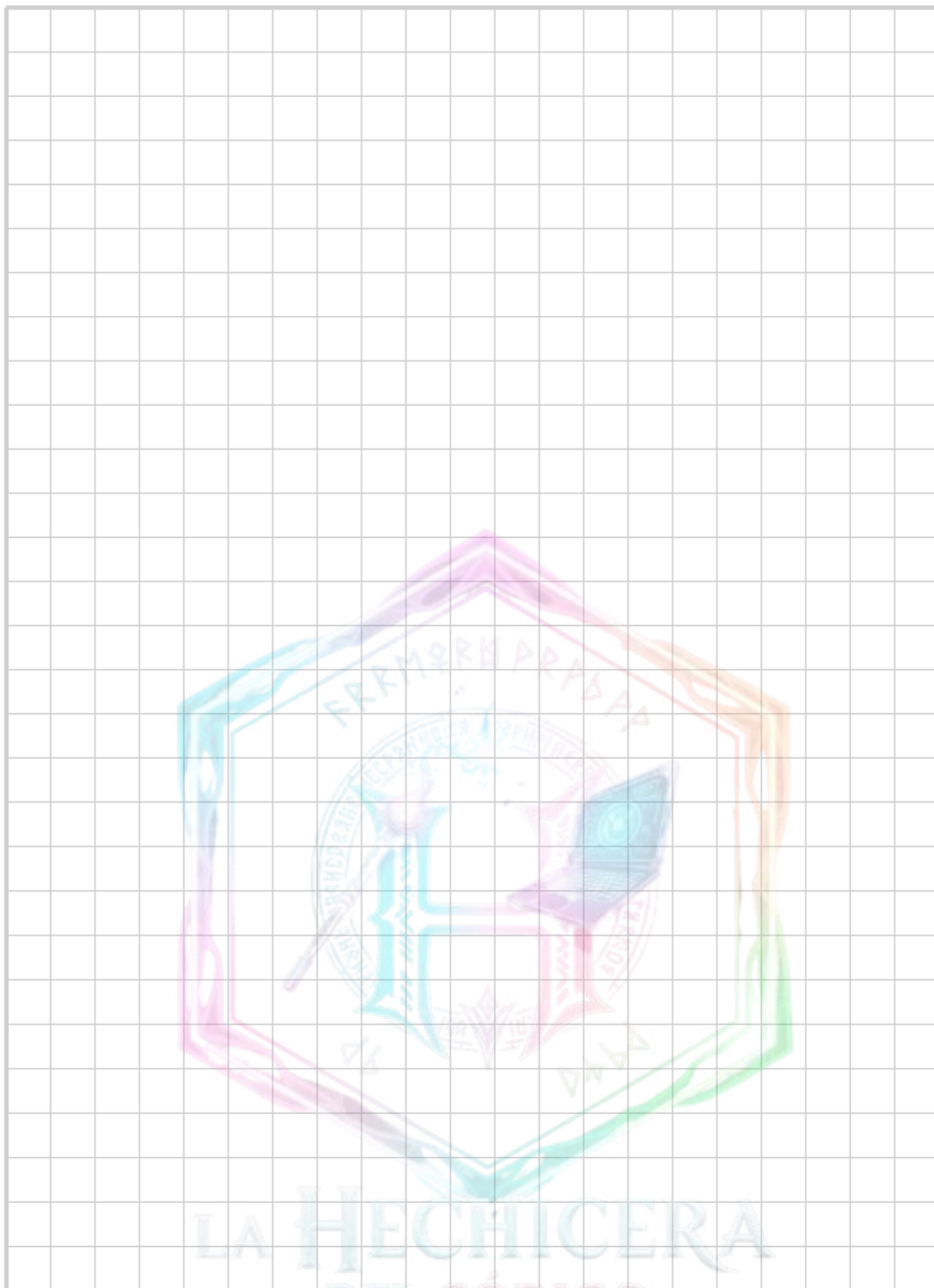
La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





PYTHON

La hechicera del código





La hechicera del código

Isela Data Maven

Te espero en otras Redes
sociales:



LA HECHICERA
DEL CÓDIGO

