Maestría en Inteligencia Artificial y Ciencia de Datos

Universidad Casa Grande

PARADIGMAS DE PROGRAMACIÓN PARA INTELIGENCIA ARTIFICIAL Y ANÁLISIS DE DATOS

INTRODUCCIÓN Y FUNDAMENTOS DE PYTHON Y R

Eduardo Javier Yepez Montenegro

25 Junio 2025

Clase 1: Introducción

Agenda

1 Introducción a los Paradigmas de Programación

1. Concepto de Paradigmas de Programación
2. Aplicación de Paradigmas en IA

Objetivos de la Clase

¿Qué aprenderemos en esta clase?

1. Conocer los principales paradigmas de programación y sus diferencias.
2. Entender cómo se aplican los paradigmas en Inteligencia Artificial y Ciencia de Datos.
3. Reconocer por qué es importante elegir el paradigma adecuado para desarrollar algoritmos en IA.

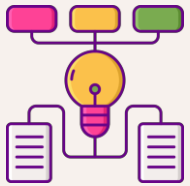
Introducción a los Paradigmas de Programación

¿Qué es?

Concepto: Un paradigma de programación es un estilo o enfoque de programación que define cómo resolver problemas a través del código.

¿Por qué es útil?

Propósito: Los paradigmas ayudan a estructurar el pensamiento del programador, facilitando la implementación y optimización del código.



¿Dónde se aplica?

Contexto: En IA y Ciencia de Datos, la elección del paradigma adecuado puede mejorar el rendimiento y escalabilidad de los algoritmos.

Referencia:Sebesta, R. W. (2016). Concepts of Programming Languages (11th Edition). Pearson.
Van Roy, P., & Haridi, S. (2004). Concepts, Techniques, and Models of Computer Programming. MIT Press.

Definición y Comparación de Paradigmas de Programación

1. Funcional
   - Se basa en funciones matemáticas puras, evitando el uso de estados mutables y efectos secundarios.
   - Características: Inmutabilidad, funciones de orden superior, recursión.
   - Lenguajes: Haskell, Lisp, Scala.

2. Orientado a Objetos (OOP)
   - Enfocado en organizar el código en objetos que encapsulan datos y comportamientos.
   - Características: Herencia, polimorfismo, encapsulación, abstracción.
   - Lenguajes: Java, Python, C++.

Referencia:Sebesta, R. W. (2016). Concepts of Programming Languages (11th Edition). Pearson.
Van Roy, P., & Haridi, S. (2004). Concepts, Techniques, and Models of Computer Programming. MIT Press.

Definición y Comparación de Paradigmas de Programación

3. Declarativo
   - Describe qué se debe lograr en lugar de como lograrlo, enfocándose en expresiones y declaraciones.
   - Características: Evaluación no secuencial, mayor abstracción, independencia del control de flujo.
   - Lenguajes: SQL, Prolog.

4. Imperativo
   - Enfocado en describir los pasos necesarios para resolver un problema.
   - Características: Control explícito del flujo de ejecución, mutabilidad.
   - Lenguajes: C, Python (cuando se usa imperativamente).

Referencia:Sebesta, R. W. (2016). Concepts of Programming Languages (11th Edition). Pearson.
Van Roy, P., & Haridi, S. (2004). Concepts, Techniques, and Models of Computer Programming. MIT Press.

Definición y Comparación de Paradigmas de Programación

Comparación:

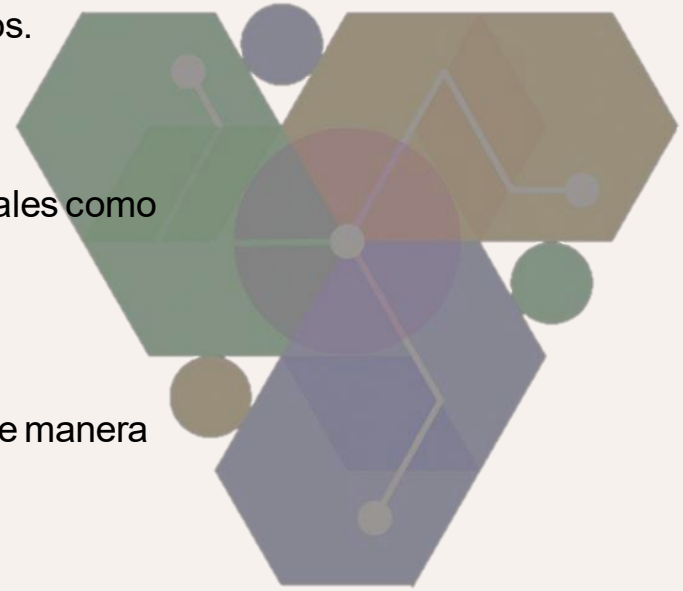
Paradigma	Forma de trabajar	Ejemplo de Lenguajes	Beneficios	Desventajas
Funcional	Funciones puras	Haskell, Lisp	Código más limpio	Curva de aprendizaje alta
Orientado a Objetos	Objetos	Java, Python	Reusabilidad	Puede ser complejo
Declarativo	Declaraciones	SQL, Prolog	Simplicidad	Limitado a casos específicos
Imperativo	Pasos explícitos	C, Python	Flexibilidad	Propenso a errores humanos

Sestoft, P. (2017). Programming Language Concepts. Springer.
Abelson, H., & Sussman, G. J. (1996). Structure and Interpretation of Computer Programs. MIT Press.

Aplicación de Paradigmas en Inteligencia Artificial (IA)

Importancia de cada paradigma en IA y Ciencia de Datos:

Paradigma Funcional:
Ideal para IA y Ciencia de Datos porque permite trabajar con funciones inmutables y datos en paralelo.
Ejemplo: TensorFlow y PyTorch utilizan conceptos funcionales para optimizar el procesamiento de datos.



Paradigma Orientado a Objetos:
Facilita el modelado de datos complejos y sistemas de IA a través de la encapsulación de objetos.
Ejemplo: En aprendizaje profundo, la orientación a objetos es útil para crear y manipular redes neuronales como objetos reutilizables en frameworks como Keras.

Paradigma Declarativo:
Útil en IA para crear modelos basados en reglas, como en la programación lógica y Prolog.
Ejemplo: En Machine Learning, lenguajes como SQL permiten procesar grandes volúmenes de datos de manera eficiente.

Paradigma Imperativo:
Proporciona control detallado en algoritmos específicos de IA, como en simulaciones y programación de bajo nivel.
Ejemplo: En el preprocesamiento de datos, donde se requiere manipulación detallada de los datos y control de flujo.

Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson.
Chollet, F. (2017). Deep Learning with Python. Manning Publications.

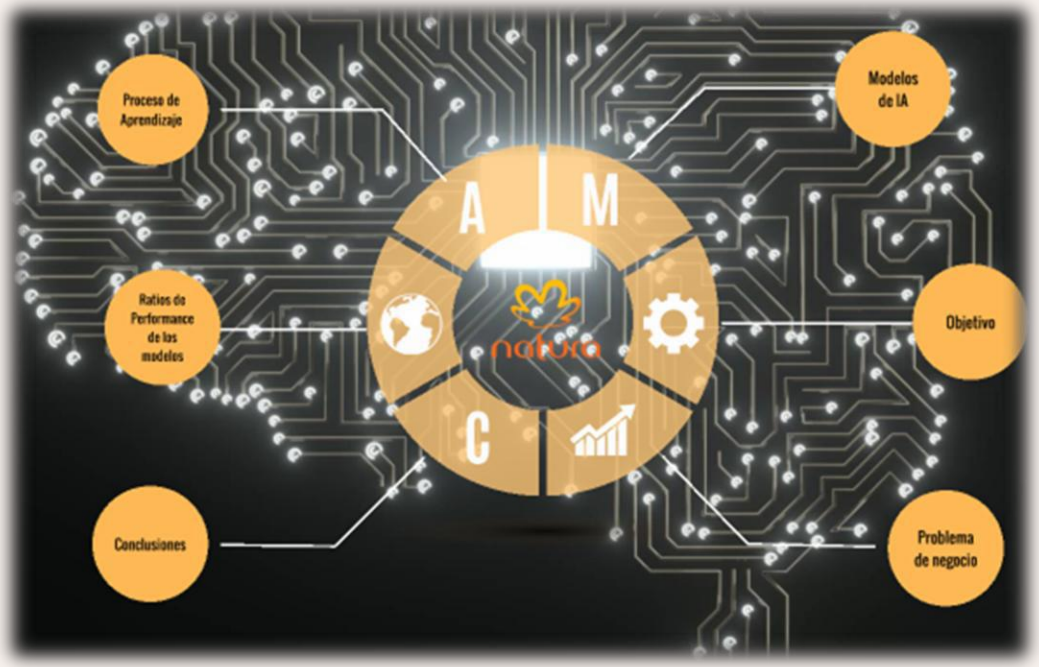
Ejemplos de Algoritmos de IA y Paradigmas

Algoritmos funcionales:
Utilizados en procesamiento paralelo y sistemas distribuidos.
Ejemplo: MapReduce.

Algoritmos orientados a objetos:
Útiles en sistemas de aprendizaje por refuerzo y redes neuronales.
Ejemplo: Diseño de agentes de IA como objetos en simulaciones.

Algoritmos declarativos:
Aplicables en IA basada en reglas y sistemas expertos.
Ejemplo: Prolog en sistemas de diagnóstico médico.

Algoritmos imperativos:
Adecuados para operaciones matemáticas intensivas.
Ejemplo: Algoritmos de optimización como gradiente descendente.



Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

Ventajas y Desventajas en IA y Ciencia de Datos

Paradigma	Ventajas para IA	Desventajas para IA
Funcional	Facilita el procesamiento en paralelo	Curva de aprendizaje compleja
Orientado a Objetos	Mejora la modularidad	Puede ser menos eficiente en procesamiento intensivo
Declarativo	Simplicidad y legibilidad	Limitado en casos complejos
Imperativo	Control total sobre el flujo	Requiere más líneas de código

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

Conclusiones

- Elegir el Paradigma Adecuado: La elección del paradigma debe basarse en el tipo de problema y la estructura de datos.
- Popularidad de Funcional y Orientado a Objetos: Estos paradigmas son comunes en IA por su eficiencia en procesamiento paralelo y modularidad.
- Beneficios para Profesionales de IA: Conocer los paradigmas ayuda a optimizar el diseño y rendimiento de los algoritmos.

Turing, A. M. (1950). Computing Machinery and Intelligence. Mind.

Gracias por su atención

Clase 1: Fundamentos de Python y R

Agenda

2 Fundamentos de Python y R

1. Fundamentos de Python para IA y Ciencia de Datos
2. Fundamentos de R para Ciencia de Datos
3. Comparación de Python y R

🎯 Objetivos de la Clase

¿Qué aprenderemos en esta clase?

1. Familiarizarse con la sintaxis básica de Python y sus bibliotecas principales para manipulación y visualización de datos.
2. Aprender a utilizar las estructuras de datos y bibliotecas clave en R para análisis estadístico y preparación de datos.
3. Comprender las diferencias y similitudes entre Python y R para saber cuándo emplear cada lenguaje en proyectos de IA y Ciencia de Datos.

Introducción a Python para IA y Ciencia de Datos

Python

Es un lenguaje de propósito general que se ha convertido en el estándar para proyectos de IA y Ciencia de Datos debido a su simplicidad y versatilidad.

Popular en la industria por:

- Su sintaxis legible.
- Amplia comunidad y soporte.
- Gran cantidad de bibliotecas y frameworks especializados.

Aplicación en IA y Ciencia de Datos: Python es ampliamente utilizado para desarrollar algoritmos de aprendizaje automático, procesamiento de datos y visualización.

Estructuras de Datos en Python

Listas: Colección ordenada y mutable. Útil para almacenar secuencias de elementos.

Ejemplo: mi_lista = [1, 2, 3]

Tuplas: Inmutables, ideales para datos que no deben cambiar.

Ejemplo: mi_tupla = (1, 2, 3)

Diccionarios: Almacenan pares clave-valor, útiles para datos asociados.

Ejemplo: mi_diccionario = {"clave1": "valor1", "clave2": "valor2"}

Aplicación en IA y Ciencia de Datos: Las estructuras de datos permiten organizar grandes cantidades de información de forma que sea fácil de procesar en análisis y modelos.

Estructuras de Datos: El Camino "Pythonico"

```
# Modo tradicional (funciona, pero es verboso)
cuadrados = []
for i in range(5):
    cuadrados.append(i**2)

# Modo Pythonico (conciso, legible y más rápido)
cuadrados_pro = [i**2 for i in range(5)]
```

Estructuras de Datos: El Camino "Pythonico"

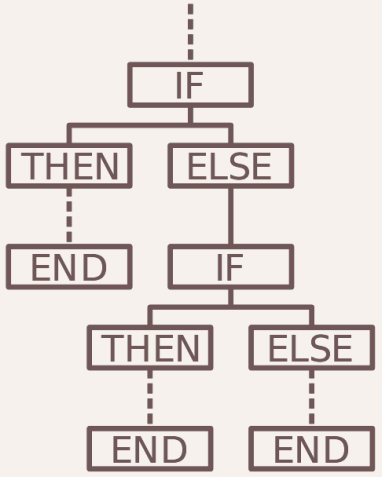
```
config = {'lr': 0.01}


# Inseguro: si la clave no existe, el programa se detiene
# batch = config['batch_size'] # Lanza KeyError!


# Profesional: usa .get() con un valor por defecto
batch = config.get('batch_size', 32) # Elegante y seguro
```

Control de Flujo en Python (Condicionales y Bucles)

Condicionales (if, elif, else): Permiten ejecutar código basado en condiciones.

Ejemplo

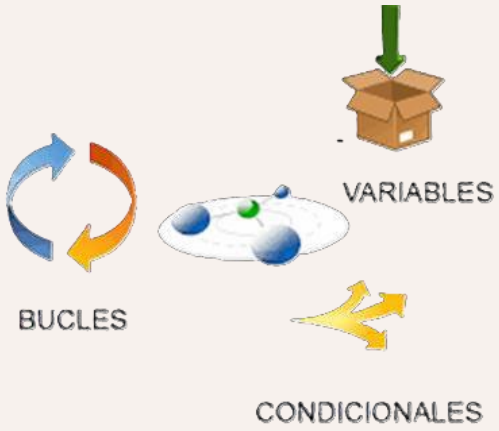
```
if x > 10:
    print("Mayor a 10")
elif x == 10:
    print("Igual a 10")
else:
    print("Menor a 10")
```



Bucles (for, while): Repetición de tareas.

Ejemplo de for:

```
for i in range(5):
    print(i)
```



Aplicación en IA y Ciencia de Datos: Permiten procesar grandes conjuntos de datos de manera automática, una habilidad esencial en modelos de Machine Learning

Funciones en Python

Definir funciones: Nos permite encapsular código para ser reutilizable y modular.

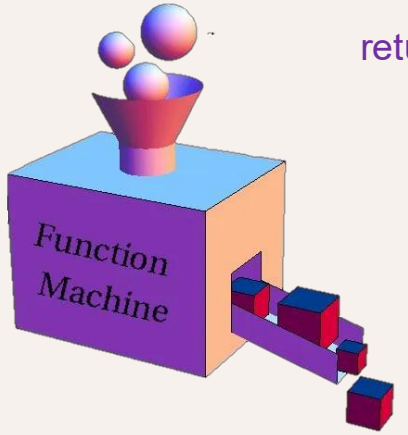
Ejemplo

```
def saludo(nombre):
    print(f"Hola, {nombre}")
```

Parámetros y retorno: Las funciones pueden recibir parámetros y devolver valores.

Ejemplo

```
def suma(a, b):
    return a + b
```



Aplicación en IA y Ciencia de Datos: Permiten crear módulos específicos para cálculos o procesamiento de datos en modelos de IA, mejorando la organización y eficiencia del código.

Funciones: Creando Herramientas Flexibles

```
# Esta función puede recibir cualquier número de argumentos
# posicionales (args) y de palabra clave (kwargs)
def funcion_flexible(*args, **kwargs):
    print(f"Argumentos posicionales: {args}")
    print(f"Argumentos de palabra clave: {kwargs}")


funcion_flexible(1, 'hola', True, modelo='ResNet', epochs=20)
# Salida:
# Argumentos posicionales: (1, 'hola', True)
# Argumentos de palabra clave: {'modelo': 'ResNet', 'epochs': 20}
```

Así es como librerías como scikit-learn o Matplotlib permiten pasar docenas de parámetros opcionales. Es clave para crear APIs robustas.

Bibliotecas Esenciales en Python para IA y Ciencia de Datos

NumPy: Manipulación de arrays de gran tamaño. Es la base para otras bibliotecas.

```
import numpy as np

arr = np.array([1, 2, 3])
```

Pandas: Manipulación y análisis de datos en formato de tablas.

```
import pandas as pd

df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
```

Matplotlib y Plotly: Visualización de datos.

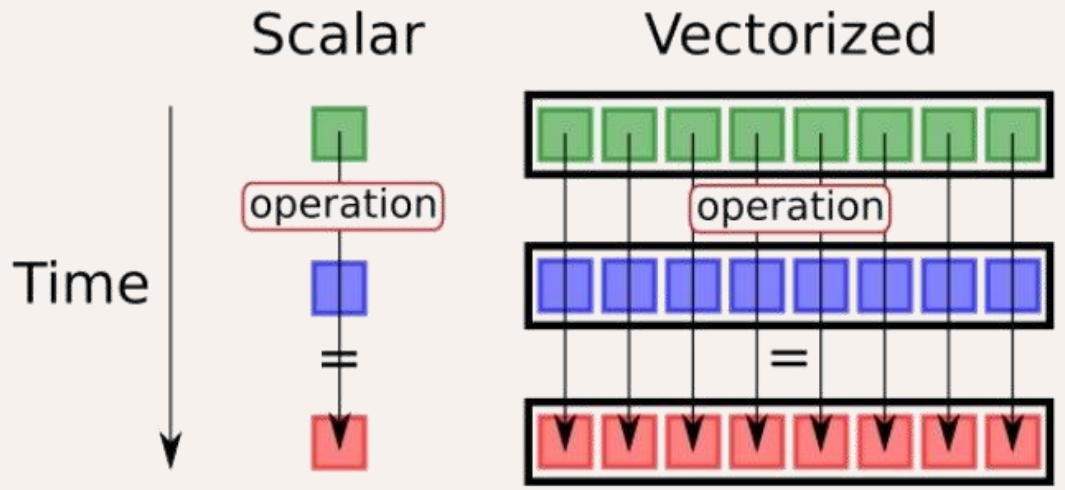
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], [4, 5, 6])plt.show()
```

Scikit-Learn: Implementación de algoritmos de Machine Learning.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

El Concepto MÁS Importante: Vectorización

La magia de NumPy/Pandas no está en los arrays, sino en la vectorización. Las operaciones se ejecutan en código C/Fortran compilado, no en bucles lentos de Python. La diferencia de velocidad es de órdenes de magnitud.



Comparison between scalar (classical) computation and vectorization.

https://medium.com/@mflova/making-python-extremely-fast-with-numba-advanced-deep-dive-2-3-f809b43f8300

Moraleja: Si estás usando un bucle for en tus datos, probablemente hay una forma vectorizada (y 100x más rápida) de hacerlo.

Manipulación de Datos con Pandas

DataFrames: Estructura de datos tabular en Pandas.

```
import pandas as pd

datos = {'Nombre': ['Carlos', 'Ana'], 'Edad': [30, 25]}

df = pd.DataFrame(datos)
```

Operaciones básicas:

Selección de columnas:

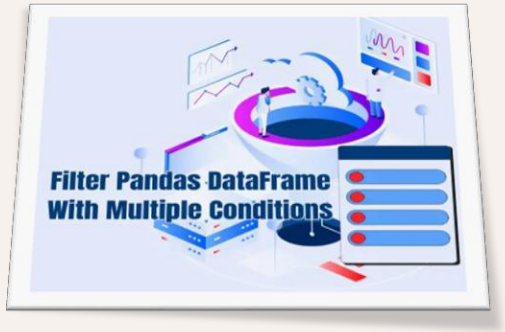
```
df['Nombre']
```

Filtro de filas:

```
df[df['Edad'] > 26]
```

Agrupación:

```
df.groupby('Edad').mean()
```



Aplicación en IA y Ciencia de Datos: Pandas permite limpiar y transformar datos en el formato adecuado para aplicar modelos de IA, siendo una herramienta fundamental en la fase de preprocesamiento.

Introducción a R para Ciencia de Datos

R

R es un lenguaje de programación especializado en estadística y análisis de datos.

Amplia adopción en la academia y la investigación debido a sus capacidades estadísticas y gráficas.

Fortalezas:

- Herramientas integradas para análisis estadístico.
- Amplias bibliotecas para manipulación de datos y visualización.
- Comunidad activa en Ciencia de Datos y estadística.

Aplicación en IA y Ciencia de Datos: R es ideal para proyectos que requieren análisis estadísticos profundos y visualización avanzada de datos.

Estructuras de Datos en R

Vectores: Colección de elementos del mismo tipo.

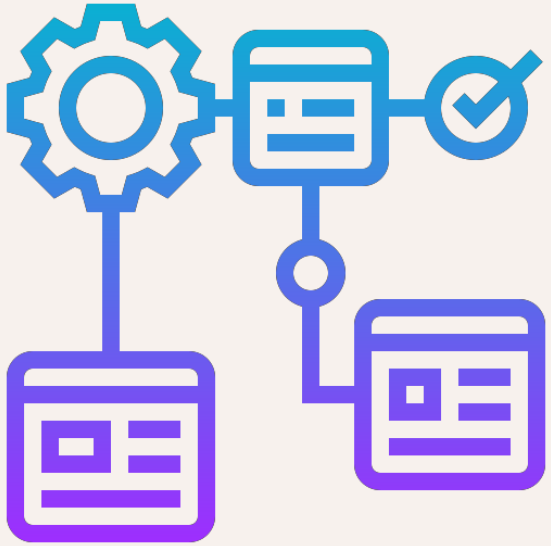
Ejemplo: mi_vector <- c(1, 2, 3)

Listas: Colección de elementos de distintos tipos.

Ejemplo: mi_lista <- list(nombre = "Carlos", edad = 30)

Data Frames: Estructura de datos tabular, similar a una hoja de cálculo.

Ejemplo: datos <- data.frame(Nombre = c("Carlos", "Ana"), Edad = c(30, 25))



Aplicación en IA y Ciencia de Datos: Estas estructuras permiten almacenar y organizar datos en R, facilitando su manipulación y análisis.

Control de Flujo en R (Condicionales y Bucles)

Condicionales (if, else):

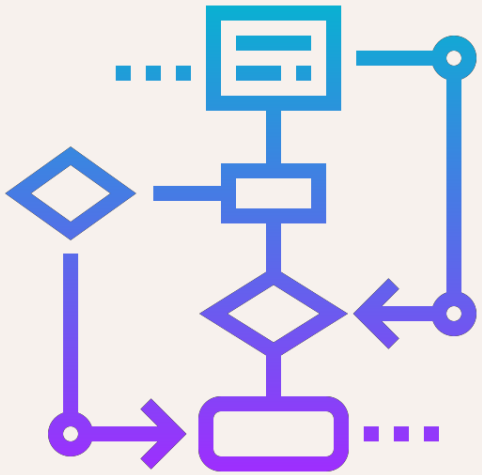
Ejecutan código basado en condiciones

```
x <- 10
if (x > 10) {
print("Mayor a 10")
} else if (x == 10) {
 print("Igual a 10")
} else {
print("Menor a 10")
}
```

Bucles (for, while):

Repetición de tareas

```
for (i in 1:5) {
 print(i)
}
```

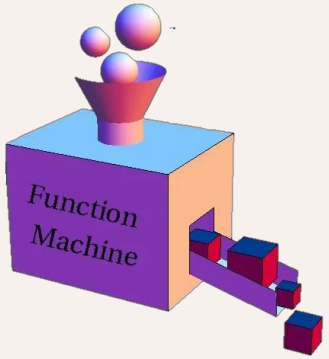


Aplicación en IA y Ciencia de Datos: Los condicionales y bucles permiten automatizar tareas en el análisis de datos, como en limpieza y transformación de datos.

Creación de Funciones en R

Definir funciones: Crear bloques de código reutilizables

```
saludo <- function(nombre){
    print(paste("Hola,", nombre))
}
```



Parámetros y retorno: Las funciones en R pueden recibir argumentos y devolver valores.

```
suma <- function(a, b){
    return(a + b)
}
```

Aplicación en IA y Ciencia de Datos: Las funciones permiten modular el código, lo que es clave en procesos de análisis complejos o repetitivos.

Bibliotecas Esenciales en R para Ciencia de Datos

Tidyverse: Conjunto de paquetes para manipulación y visualización de datos.

```
library(tidyverse)
datos <- tibble(Nombre = c("Carlos", "Ana"), Edad = c(30, 25))
```

ggplot2: Paquete de visualización basado en gramática de gráficos.

```
library(ggplot2)

ggplot(datos, aes(x = Nombre, y = Edad)) + geom_bar(stat = "identity")
```

dplyr: Paquete para manipulación de datos.

```
library(dplyr)

datos %>% filter(Edad > 25)
```

Manipulación de Datos con dplyr y tidyverse

Operaciones comunes en dplyr:

Seleccionar columnas: select(datos, Nombre)

Filtrar filas: filter(datos, Edad > 25)

Agrupar y resumir: group_by(datos, Edad) %>% summarise(count = n())

Pipelines (%>%): Permiten aplicar múltiples transformaciones en secuencia.

Ejemplo:

```
datos %>%
  filter(Edad > 25) %>%
  summarise(promedio_edad = mean(Edad))
```

Aplicación en Ciencia de Datos: La manipulación de datos es fundamental para preparar la información antes de aplicar análisis o modelos de IA, y dplyr facilita estas tareas en R.

Comparación de Python y R

Definición de Variables:

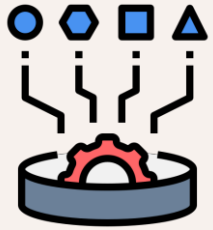
- **Python:** x = 5
- **R:** x <- 5

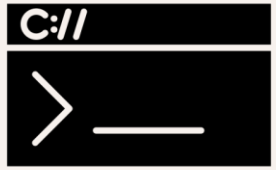
Impresión en Consola:

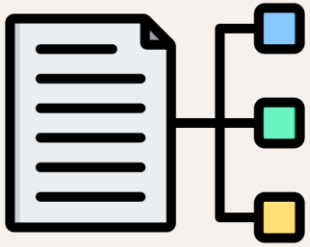
- **Python:** print("Hola, Mundo")
- **R:** print("Hola, Mundo")

Estructuras de Datos:

- **Listas en Python:** [1, 2, 3]
- **Vectores en R:** c(1, 2, 3)







Manipulación de Datos en Python y R

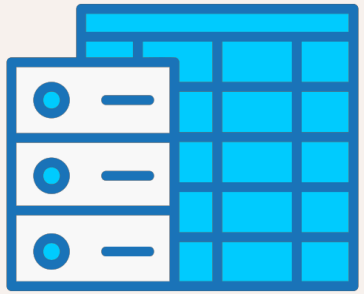
Data Frames:

Python (usando pandas):

```
import pandas as pd

datos = pd.DataFrame({'Nombre': ['Carlos', 'Ana'], 'Edad': [30, 25]})
```

R:

```
datos <- data.frame(Nombre = c("Carlos", "Ana"), Edad = c(30, 25))
```



Filtrado de Datos:

Python: `datos[datos['Edad'] > 25]`

R: `datos[datos$Edad > 25, ]`

Bibliotecas Clave para IA y Ciencia de Datos

Python:

- NumPy y Pandas para manipulación de datos.
- Scikit-Learn para algoritmos de Machine Learning.
- Matplotlib y Plotly para visualización.

R:

- tidyverse para manipulación de datos (incluye dplyr, tidyr).
- caret para algoritmos de Machine Learning.
- ggplot2 para visualización avanzada.

¿Cuándo usar Python o R en IA y Ciencia de Datos?

Python:

- **Desarrollo impulsado por la comunidad:** Python es mantenido por una comunidad diversa de desarrolladores, lo que puede incluir enfoques menos formales en ciertas herramientas o librerías, aunque son ampliamente adoptadas.
- **Versatilidad:** Ideal para integraciones con otros sistemas, desarrollo de software y aplicaciones empresariales.
- **Machine Learning y Deep Learning:** Soporte robusto con librerías como TensorFlow y PyTorch.
- **Integración con producción:** Muy eficiente para flujos de trabajo completos.

R:

- **Respaldo estadístico:** Diseñado específicamente por estadísticos, con funciones validadas científicamente, lo que lo convierte en una herramienta de confianza para análisis rigurosos.
- **Análisis académico y científico:** Preferido en investigaciones científicas y entornos donde se requiere alta precisión estadística.
- **Visualización avanzada:** Ofrece gráficos de alta calidad con ggplot2 y otras herramientas.
- **Menos integración:** Más limitado fuera del análisis estadístico puro.

Gracias por su atención