guayerd

Fundamentos IA

Introducción IA y datos

Clase 4

En colaboración con

IBM **SkillsBuild**

¡Bienvenidos!

¿Nos presentamos?

- ¿Qué recuerdan de la clase anterior?
- ¿Qué esperan aprender?
- ¿Tienen alguna pregunta?

guayerd

En colaboración con

IBM SkillsBuild

Contenidos

Por temas

01
- Introducción IA

02
- Fundamentos del dato
- Pensamiento computacional

03
- Introducción Python

04
- Introducción a Python

guayerd

En colaboración con

IBM **SkillsBuild**

Objetivos de la clase

- Estructuras de control
- Funciones
- Numpy

guayerd

En colaboración con

IBM SkillsBuild

Introducción a la IA y los datos

Introducción a Python

guayerd

En colaboración con

IBM SkillsBuild

Plataforma Skill Build: Python

guayerd

En colaboración con
IBM SkillsBuild

El arte de preguntar

- Orientan decisiones y evitan la improvisación
- Transforman datos en conocimiento útil
- Abren caminos hacia soluciones innovadoras
- Conectan problemas con objetivos claros
- Son la base de todo proceso analítico

Tipos de indagación

Explorar

¿Qué patrones existen?

Comparar

¿En qué se diferencian?

Explicar

¿Por qué sucede esto?

Anticipar

¿Qué ocurrirá después?

Estructurar cualquier análisis

Problema

- Qué [acción/resultado] queremos lograr
- Por qué [este problema] es prioritario
- Cuándo [tiempo/plazo] necesitamos respuestas

Datos

- Qué [información] tenemos disponible
- Cómo [organizar/limpiar] para analizarla
- Qué [limitaciones] debemos considerar

Enfoque

- Qué [relaciones/patrones] explorar primero
- Qué [variables] incluir o descartar
- Cómo [validar] que los resultados sean confiables

Aplicación en Machine Learning

Fórmula

[Verbo de acción] + [objeto específico] + [criterio de éxito]

Aplicación en Machine Learning

Ejemplos:

- ¿Predecir qué clientes abandonarán con +85% precisión?
- ¿Clasificar productos defectuosos minimizando falsos negativos?

Preguntas de validación:

- ¿Los datos representan el problema real?
- ¿El modelo mejora la decisión actual?
- ¿Existen sesgos o consideraciones éticas?

Estructuras de control

Definen **cuándo** y **bajo qué condiciones** se ejecutan las instrucciones.

Tipos

- **Condicionales:** ejecutan código si se cumple una condición
- **Bucles:** repiten código mientras la condición sea verdadera

Son la base para que un programa deje de ser lineal y pase a ser interactivo

Condicionales

Los condicionales son bloques de código que se ejecutan únicamente si se cumple una condición. El resultado de esta condición debe ser booleano (True o False). Esto se logra mediante la sentencia **if**. Con la sentencia **elif** se puede agregar un número arbitrario de condiciones.

Por otra parte, se puede ejecutar código si la/s condición/es no se cumple/n con la sentencia **else**.

```
>>> valor = 0
>>> if (valor < 0):
>>>     print('El número es negativo')
>>> elif (valor > 0):
>>>     print('El número es positivo')
>>> else:
>>>     print('El número es igual a cero')
El número es igual a cero
```

Condicionales

Sintaxis

```
if condición:
    # código si la condición es verdadera
elif otra_condición:
    # código si la anterior fue falsa y esta es verdadera
else:
    # código si ninguna condición es verdadera
```

Ejemplo

```
edad = 20
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

Ciclos Iterativos o Loops (bucles)

Son bloques de código que se repiten una cierta cantidad de veces en función de ciertas condiciones.

Un ciclo **for** repite un bloque de código tantas veces como elementos haya dentro del rango entre 1 y 10

Un ciclo **while** repite un bloque de código mientras que cierta condición se cumpla:

```
>>> for n in range(1,10):
>>>     print(n)
1
2
3
4
5
6
7
8
9
```

```
>>> n = 1
>>> while (n < 10):
>>>     print(n)
>>>     n = n + 1
1
2
3
4
5
6
7
8
9
```

Bucles

For

Recorre elementos de una secuencia (listas, cadenas, rangos, etc.)

```
for variable in secuencia:
    # código
```

While

Repite mientras una condición sea verdadera

```
while condición:
    # código
```

Sentencia Break

La sentencia break permite alterar el comportamiento de los bucles while y for. Concretamente, permite terminar con la ejecución del bucle. Esto significa que una vez se encuentra la palabra break, el bucle se habrá terminado.

El range(5) generaría 5 iteraciones, donde la i valdría de 0 a 4. Sin embargo, en la primera iteración, terminamos el bucle prematuramente. El break hace que nada más empezar el bucle, se rompa y se salga sin haber hecho nada.

```
>>> for i in range(5):
>>>   print(i)
>>>   break
0
```

Un ejemplo un poco más útil, sería el de buscar una letra en una palabra. Se itera toda la palabra y en el momento en el que se encuentra la letra que buscábamos, se rompe el bucle y se sale.

```
>>> cadena = 'Python'
>>> for letra in cadena:
>>>     if letra == 'h':
>>>         print("Se encontró la h")
>>>         break
>>>     print(letra)
P
y
t
Se encontró la h
```

El break también nos permite alterar el comportamiento del while. En el ejemplo, la condición while True haría que la sección de código se ejecutará indefinidamente, pero al hacer uso del break, el bucle se romperá cuando x valga cero.

```
>>> x = 5
>>> while True:
>>>     x -= 1
>>>     print(x)
>>>     if x == 0:
>>>         break
>>> print("Fin del bucle")
4
3
2
1
0
Fin del bucle
```

Por norma general, y salvo casos muy concretos, si hay un while True, es probable que haya un break dentro del bucle.

Bucles

Control

break: interrumpe el bucle

```
for i in range(5):
        if i ==3:
            break
```

continue: salta a la siguiente iteración

```
for i in range(5):
        if i ==2:
            continue
```

pass: no hace nada (placeholder)

```
for i in range(3):
        pass
```

range() genera secuencias

Bucles

Patrones de iteración

Estrategias para recorrer y procesar elementos en una secuencia.

Acumulador: sumar o multiplicar valores

```
suma = 0
for x in numeros:
        suma += x
```

Contador: contar elementos que cumplen condición

```
contar = 0
for x in numeros:
        if x > 10:
                contar += 1
```

Buscador: encontrar un elemento específico

```
for x in numeros:
        if x == 15:
                print("Encontrado")
                break
```

Filtro: procesar sólo ciertos elementos

```
for x in numeros:
        if x % 2 ==0:
                print(x)
```

Sentencia Continue

El uso de continue al igual que el ya visto break, permite modificar el comportamiento de los bucles while y for.

```
>>> cadena = 'Python'
>>> for letra in cadena:
>>>     if letra == 'P':
>>>         continue
>>>     print(letra)
y
t
h
o
n
```

Funciones

Las funciones son una secuencia de comandos que ejecutan una sección de código.

En Python las funciones se definen usando la palabra reservada **def** y luego el nombre de la función con paréntesis y dos puntos que indican que lo que sigue son las sentencias, eventualmente una función debe retornar un valor, para esto se usa la palabra reservada **return**.

Funciones

Ventajas

- Evitan repetir código
- Facilitan el mantenimiento
- Mejoran la organización del programa

Sintaxis

```
# Definir

def nombre_funcion(parametros):
        # código
        return resultado # opcional

# Llamar

nombre_funcion(parametros)
```

Funciones

Parámetro y retorno

Las funciones pueden recibir datos (parámetros) y devolver resultados con return.

Ejemplo

```
def cuadrado(x):
    return x * x

resultado = cuadrado(5)
```

Extra:
Las variables dentro de una función son locales, mientras que las de afuera son globales

Límites al declarar funciones

- Los nombres no pueden comenzar con dígitos
- No pueden utilizar una palabra reservada
- Las variables deben tener diferentes nombres
- Los nombres de las funciones deben ser descriptivas de lo que hacen las funciones "Imprimir_valor_variable"
- Los parámetros pueden tener valores por defecto.
- Puede devolver ningún, uno o más de un valor, y de diferentes tipos de datos.
- Se puede asignar a variables, el resultado de retorno de una función.

Declarar y Ejecutar

Declarar una función es escribir su estructura y ejecutar una función es llamar la función y ejecutar su código, estas dos cosas ocurren en sentencias diferentes.

Para devolver dos números ordenados de menor a mayor podemos hacerlo así:

```
>>> def ordenar_dos_numeros(num1=0, num2=0):
>>>     if (num1 > num2):
>>>         return num2, num1
>>>     else:
>>>         return num1, num2
```

Funciones

Integradas

Categoría	Función	Descripción	Ejemplo
E/S	print()	Mostrar en pantalla	print("Hola")
	input()	Recibir datos	nombre = input("Nombre: ")
Conversión	int()	Convertir a entero	int("5") → 5
	float()	Convertir a decimal	float("3.14") → 3.14
	str()	Convertir a texto	str(10) → "10"
Matemáticas	abs()	Valor absoluto	abs(-5) → 5
	round()	Redondear	round(3.7) → 4
	min() / max()	Mínimo y máximo	max(2,5,1) → 5
	sum()	Suma lista	sum([1,2,3]) → 6
Colecciones	len()	Longitud	len("Hola") → 4
	range()	Secuencia números	range(5) → 0,1,2,3,4
	sorted()	Ordenar	sorted([3,1,2]) → [1,2,3]

Registro de temperaturas

1. Solicitar la temperatura de 5 días consecutivos
2. Mostrar la temperatura promedio, máxima y mínima de la semana
3. Contar cuántos días tuvieron temperatura alta (mayor a 25°C)
4. Mostrar un resumen completo de los datos registrados

¿Qué son las librerías?

Conjuntos de herramientas que **amplían las capacidades de Python**, ya que el lenguaje sólo incluye recursos básicos.

Ventajas

- Permiten trabajar eficientemente con grandes volúmenes de datos.
- Evitan tener que programar soluciones que ya están optimizadas.
- Aprovechan años de desarrollo y conocimiento especializado.

NumPy

¿Qué es?

NumPy (Numerical Python) es una librería numérica de Python.

Es base de todos los cálculos científicos.

Es de código abierto, proporciona estructuras de datos matriciales y funciones matemáticas de alto nivel.

NumPy
(Numerical Python)

Librería esencial para el **trabajo con datos numéricos** en Python.

- Procesamiento eficiente de arrays multidimensionales
- Operaciones matemáticas y estadísticas optimizadas
- Base fundamental de librerías como Pandas y Scikit-learn
- Rendimiento superior para cálculos numéricos masivos

Comandos

- **Instalación:** pip install numpy
- **Uso:** import numpy as np

Accede a funciones con alias.nombre_funcion()

Arrays en NumPy

Estructura de datos optimizada para cálculos numéricos, similar a las listas pero con mayor eficiencia.

- Almacena elementos del mismo tipo de dato
- Soporta operaciones en una y múltiples dimensiones
- Permite operaciones matemáticas vectorizadas
- Procesamiento simultáneo de todos los elementos

Velocidad de procesamiento hasta 100 veces superior a listas Python tradicionales.

¿Qué son?

En **NumPy** se trabaja con una estructura de datos llamada array o arreglos numéricos multidimensionales.

✔ Parecidos a las listas de Python, heredan algunas propiedades como el ser mutables y poder realizar slicing.

✘ Tienen diferencias importantes: son menos pesados, más rápidos y permiten crear fácilmente arrays de N dimensiones.

Tipos de arrays

- Un array unidimensional puede ser una fila o una columna de una tabla, igual que una lista, esta se conoce como vector.
- Un array bidimensional es lo que llamamos comúnmente matriz.
- Un array de 3 dimensiones o más, es decir, una matriz de matrices, se conoce como tensor.

Sintaxis básica

De arrays

Importación y creación

```
import numpy as np
vector = np.array([elementos])
matriz = np.array([[fila1], [fila2]])
```

Acceso

```
array[índice]        # Elemento específico
array[inicio:fin]    # Rango de elementos
```

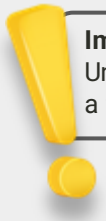
Modificación

```
array[índice] = valor   # Cambiar elemento
array[:] = valor        # Cambiar todos los elementos
```

shape muestra dimensiones

Operaciones vectorizadas

- **Array con escalar:** array + número, array * número, array ** número
- **Array con array:** array1 + array2, array1 > array2 (mismo tamaño)
- **Funciones universales:** np.sqrt(), np.sin(), np.abs(), np.exp(), np.log()
- **Operaciones de comparación:** array > valor (genera array booleano)
- **Operaciones lógicas:** array1 & array2, array1 | array2



Importante:
Una operación se aplica automáticamente a todos los elementos del array

Arrays especiales

NumPy ofrece **funciones para crear arrays con patrones específicos** sin definir elementos manualmente.

- **Valores constantes:** np.zeros(), np.ones(), np.full()
- **Secuencias numéricas:** np.arange(), np.linspace()
- **Aleatorios:** np.random.rand(), np.random.randint()
- **Estructura:** np.eye(), np.empty()

Propiedades básica

De arrays

Comandos esenciales para **inspeccionar y entender la estructura de arrays.**

Información dimensional

- **array.shape:** dimensiones del array
- **array.ndim:** número total de dimensiones
- **array.size:** cantidad total de elementos

Información del contenido

- **array.dtype:** tipo de datos almacenados
- **array.itemsize:** tamaño en bytes de cada elemento
- **array.nbytes:** memoria total ocupada

Los arrays son la base de toda la inteligencia artificial moderna

Crear arrays

A partir de una lista

```
>>> my_list = [0, 1, 2, 3, 4]
>>> print(np.array(my_list))
[0 1 2 3 4]
```

A partir de secuencias

```
>>> print(np.arange(start=2, stop=10, step=2))
[2 4 6 8]
```

```
>>> print(np.linspace(0, 1, 11))
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

Crear arrays

Predefinidos

```
>>> print(np.zeros(4))
[0. 0. 0. 0.]
```

```
>>> print(np.ones(6))
[1. 1. 1. 1. 1. 1.]
```

```
>>> print(np.full(shape=(2, 2), fill_value=5))
[[5 5]
 [5 5]]
```

```
>>> base = np.linspace(2, 6, 4)
>>> print(np.full_like(base, np.pi))
[3.14159265 3.14159265 3.14159265 3.14159265]
```

Crear arrays

Aleatorios

```
>>> print(np.random.rand(2, 2))
[[0.62740202 0.11171536]
 [0.47526728 0.19739417]]
>>>
>>> print(np.random.uniform(low=0, high=1, size=6))
[0.7878737  0.3431897  0.77765595 0.60943181 0.30961326 0.60167083]
 >>>
>>> print(np.random.randn(2, 2))
[[ 0.91140011  1.72792052]
 [-0.84028707 -0.27378577]]
>>>
>>> print(np.random.normal(loc=0, scale=2, size=6))
[-2.36743682 -3.12673482 -1.14254395 -3.19805542 -1.11930443 -2.70161226]
```

Slicing con arrays

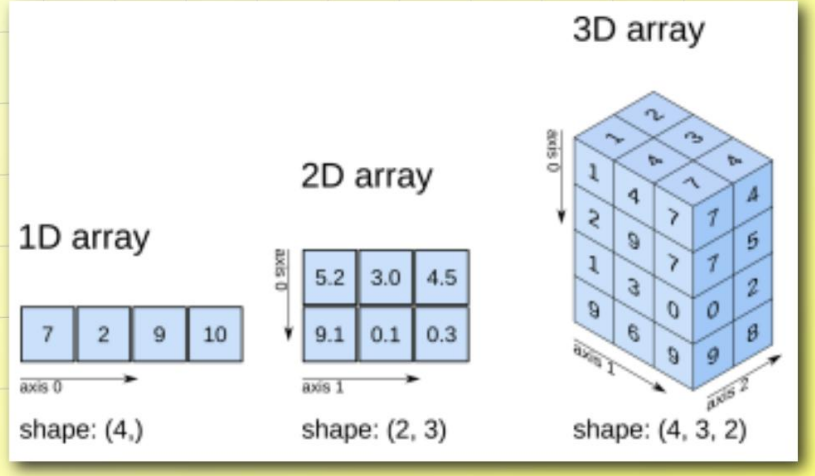
```
>>> matrix_cool = np.arange(9).reshape(3, 3)
>>> print(matrix_cool)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> print(matrix_cool[1, 2])
5
>>> print(matrix_cool[0, :])
[0 1 2]
```

```
>>> print(matrix_cool[:, 1])
[1 4 7]
>>> print(matrix_cool[:, 1:])
[[1 2]
 [4 5]
 [7 8]]
>>> print(matrix_cool[0:2, 0:2])
[[0 1]
 [3 4]]
```

Tamaño de arrays

```
>>> B = np.reshape(a, [3,3])
>>> print(B)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
>>> a = np.arange(1,10)
>>> print(a)
[1 2 3 4 5 6 7 8 9]
```



Copiar arrays

```
>>> a1 = np.array([2, 4, 6])
>>> a2 = a1.copy()
>>> a1[0] = 8
>>> print(a1)
>>> print(a2)
[8 4 6]
[2 4 6]
```

Operaciones con arrays

Suma

```
>>> A = np.arange(5, 11)
>>> print(A)
[ 5  6  7  8  9 10]
>>> print(A + 10)
[15 16 17 18 19 20]
```

Resta

```
>>> B = np.full(4, 3)
>>> C = np.ones(4, dtype='int')
>>> print(B)
[3 3 3 3]
>>> print(C)
[1 1 1 1]
>>> print(B - C)
[2 2 2 2]
```

Operaciones con arrays

Shape (forma)

```
>>> a = np.array([[2,3],[2,3],[2,3]])
>>> a.shape
(3, 2)
>>> b = np.array([[1,6,5,2,7],[1,2,7,0,9]])
>>> b.shape
(2, 5)
```

Multiplicación

```
>>> np.matmul(a,b)
array([[ 5, 18, 31,  4, 41],
       [ 5, 18, 31,  4, 41],
       [ 5, 18, 31,  4, 41]])
```

Trasposición

```
array([[ 5,  5,  5],
       [18, 18, 18],
       [31, 31, 31],
       [ 4,  4,  4],
       [41, 41, 41]])
```

Primeras ventas

Acabas de abrir tu primera tienda y quieres analizar tus primeras 10 ventas para entender el arranque del negocio.

1. Solicitar al usuario el monto de las primeras 10 ventas
2. Calcular el promedio de estas ventas iniciales
3. Identificar cuáles ventas estuvieron por encima del promedio
4. Calcular el total recaudado en estas primeras ventas
5. Determinar cuál fue tu mejor y peor venta inicial

Proyecto

Tienda Aurelion

- **Documentación:** notebook Markdown
- **Desarrollo técnico:** programa Python
- **Visualización de datos:** dashboard en Power BI
- **Presentación oral:** problema, solución y hallazgos

guayerd

En colaboración con

IBM **SkillsBuild**

Codificación

Trabajo en equipo

En relación al programa en Python que se desarrollará:

1. Crea el **diagrama de flujo**
2. Desarrolla el **programa interactivo**

Preparación demo

próxima clase

guayerd

En colaboración con

IBM **SkillsBuild**

Características

1° demo: asincrónica

- **Tema:** Inteligencia Artificial
- **Medio:** Carpeta personal en Drive
- **Fecha límite:** 05/10
- **Devolución:** En salitas por equipo

Contenido

1° demo: asincrónica

Documentación (.md)

- Tema, problema y solución
- Dataset de referencia: fuente, definición, estructura, tipos y escala
- Información, pasos, pseudocódigo y diagrama del programa
- Sugerencias y mejoras aplicadas con Copilot

Programa (.py)

- Debe permitir obtener información del proyecto

Instrucciones (.md)

- Instrucciones para Copilot

Criterios

1° demo: asincrónica

- **Tema, problema y solución** claros, vinculados a la base de datos brindada
- **Fuente, definición, estructura, tipos y escala** según lo trabajado en clase 2
- **Pasos, pseudocódigo y diagrama** que represente el desarrollo del programa en Python
- Sugerencias de **Copilot** aceptadas y descartadas
- **Programa en Python** interactivo que permita consultar la documentación de manera interactiva y sin errores de ejecución

Retro

¿Cómo nos vamos?

- ¿Qué fue lo más útil de la clase?
- ¿Qué parte te costó más?
- ¿Qué te gustaría repasar o reforzar?

guayerd

En colaboración con

IBM SkillsBuild