guayerd

Fundamentos IA

Introducción IA y datos

Clase 3

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

- Preparación del entorno
- Variables, tipos de datos y operadores
- Colecciones

guayerd

En colaboración con

IBM **SkillsBuild**

Introducción a la IA y los datos

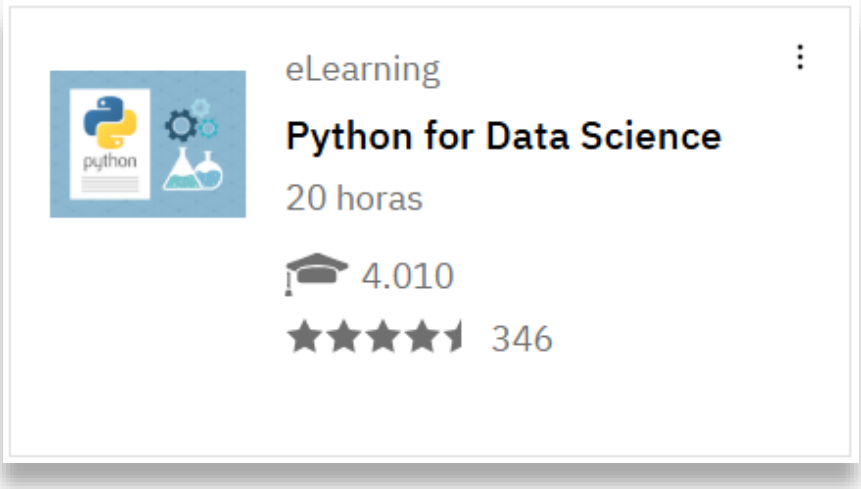
Introducción a Python

guayerd

En colaboración con

IBM **SkillsBuild**

Plataforma Skill Build: Python



guayerd

En colaboración con

IBM **SkillsBuild**

¿Qué es VS Code?

Editor de código ligero y extensible.

- **Multiplataforma:** Windows, macOS, Linux
- **Extensiones:** Python, Jupyter, Git
- **Integrado:** terminal, depuración, intelliSense

Preparar entorno

Requisitos previos

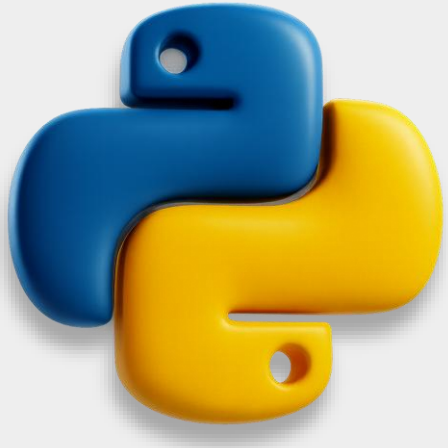
1. Descargar e instalar VS Code
2. Descargar e instalar Python

Configuración en VS Code

Instalar las extensiones del Paquete de Español, Python, Jupyter y Github Copilot desde el Marketplace.

Extra:
Permite añadir carpetas y documentos al Explorador del proyecto.

¿Qué es Python?



- **Lenguaje de programación** (.py) flexible y fácil de leer
- Destaca por su sintaxis clara, gran ecosistema y comunidad
- Se utiliza en automatización, desarrollo web, análisis de datos e inteligencia artificial

¿Qué es Jupyter?

Cuadernos interactivos (.ipynb) que permiten:

- Código paso a paso con resultados inmediatos (tablas y gráficos)
- Documentación clara con Markdown
- Aprendizaje interactivo y exploración de datos

Markdown (.md)

Conceptos esenciales

Del entorno

- **Editor (VS Code):** donde escribes y ejecutas código
- **Extensión:** complemento que añade funcionalidades
- **Intérprete:** motor que ejecuta lenguajes
- **IntelliSense:** autocompletado y ayuda contextual
- **Terminal integrada:** consola dentro del editor
- **Depurador:** herramienta para ejecutar código paso a paso
- **Script:** programa lineal en texto plano
- **Notebook:** Documento interactivo con código y resultados

Buenas prácticas

- **Organizar estructura:** Una carpeta por proyecto con nombres descriptivos
- **Configurar Auto Save:** Evita perder cambios no guardados
- **Verificar intérprete:** Seleccionar el lenguaje correcto antes de ejecutar
- **Revisar código IA:** Entender y validar sugerencias automáticas
- **Documentar código:** Comentarios claros y ejemplos prácticos

Preparación del entorno

1. Instalar Python
2. Instalar Visual Studio Code
3. Agregar la extensión de idioma en Español
4. Instalar la extensión de Python en VS Code
5. Instalar la extensión de Jupyter en VS Code
6. Instalar la extensión de Copilot en VS Code

Variables

Contenedores que **almacenan valores reutilizables**.

- **Asignación:** variable = valor
- **Características:** Tipado dinámico, modificable
- **Verificar tipo:** type(variable)

Reglas nomenclatura

- usar snake_case (palabras separadas por guiones bajos)
- Nombres descriptivos y significativos
- No empezar con números
- Sensible a mayúsculas
- No usar palabras reservadas

Tipos de datos básicos

Categorías que definen el tipo de información almacenada

- **Numérico:** int, float
- **Texto:** str
- **Booleano:** bool (true/false)
- **Ausencia de valor:** None

Python maneja tipado dinámico, lo que permite que el tipo de una variable cambie sin necesidad de declararlo previamente

Conversión de tipos

Transformar datos a otro tipo para su uso en el código.

- **Casting explícito:** tipo_destino(valor_original)
  - Ejemplo: str(3) >> “3”
- **Redondeo:** round(número, decimales)
  - Ejemplo: round(3.1416,2)

Operadores esenciales

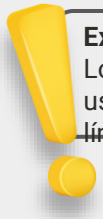
Elementos que realizan cálculos, comparaciones y evaluaciones en el código.

- **Aritmético:** +, -, *, /, //, %, **
- **Comparación:** ==, !=, <, <=, >, >=
- **Lógicos:** and, or, not
- **Asignación compuesta:** +=, -=, *=, /=
- **Pertenencia:** in, not in

Texto y formato

Herramientas para trabajar con cadenas.

- **Slicing (extraer fragmentos)**: str[inicio:fin]
- **Métodos comunes:**
  - minúsculas: .lower()
  - mayúsculas: .upper()
  - eliminar espacios: .strip()
  - reemplazar texto: .replace(viejo, nuevo)
- **Interpolación (insertar variables):** f"texto {variable}"
- **Caracteres especiales:**
  - Salto de línea: \n
  - Tabulación: \t



Extra:
Los comentarios documentan el código usando # para una línea o """ para varias líneas

Entrada y salida

Interacción con el usuario para recibir datos y mostrar resultados.

- **Entrada (solicita datos al usuario):** input("Mensaje: ")
- **Salida (muestra en pantalla):** print(valor1, valor2, ...)
  - Separador: sep
  - Final de línea: end

¿Cuántos años tienes?

Crea un programa interactivo que calcule la edad basándose en el año de nacimiento.

1. Solicita al usuario su nombre y año de nacimiento
2. Calcula la edad basándose en el año actual y muestra qué tipo de dato es
3. Convierte la entrada del usuario para poder hacer cálculos
4. Determina si la persona es mayor de edad y guarda esa información
5. Muestra un saludo personalizado con el nombre y la edad calculada

Colecciones

Estructuras que almacenan y gestionan **múltiples elementos dentro de una sola variable**.

Características

- Organizan conjuntos de información
- Admiten elementos de distintos tipos
- Las secuencia (listas/tuplas) se basan en índices
  - **índice:** posición del elemento (empieza en 0)
  - **Valor:** dato guardado en esa posición

Tipos

- **Listas (list):** ordenadas y modificables
- **Tuplas (tuple):** ordenadas e inmutables
- **Conjuntos (set):** no ordenados, sin duplicados
- **Diccionarios (dict):** pares clave-valor

Colecciones

Listas

- **Creación:** nombre_lista = [valor 1, valor 2, ...]
- **Acceso:** nombre_lista[índice]
- **Modificación:** nombre_lista[índice] = nuevo_valor
- **Ordenada:** mantienen el orden de inserción
- **Usos**
  - Guardar datos que cambian con el tiempo
  - Procesar elementos en un bucle
  - Insertar, eliminar o actualizar valores fácilmente

Slicing (sublistas)

- Extrae porciones de la lista especificando rangos
- **Sintaxis:** nombre_lista[inicio:fin:paso]
  - Inicio incluido, fin excluido
  - paso define saltos de elementos
  - [::-1] invierte la lista
  - [:] copia completa

Colecciones

Listas - métodos

Método	Descripción	Ejemplo
append(x)	Agrega elemento al final	frutas.append("uva")
insert(i, x)	Inserta en posición específica	frutas.insert(1, "kiwi")
remove(x)	Elimina primera ocurrencia del valor	frutas.remove("pera")
sort()	Ordena la lista	frutas.sort()
reverse()	Invierte el orden	frutas.reverse()
index(x)	Devuelve índice de primera ocurrencia	frutas.index("banana")
count(x)	Cuenta ocurrencias del valor	frutas.count("uva")
clear()	Elimina todos los elementos	frutas.clear()
copy()	Crea una copia de la lista	nueva = frutas.copy()

Colecciones

Tuplas

- **Creación:** nombre_tupla = (valor 1, valor 2, ...)
- **Acceso:** nombre_tupla[índice]
- **Modificación:** no se pueden cambiar (inmutables)
- **Ordenada:** mantienen el orden de inserción
- **Usos**
  - Agrupar datos que no deben cambiar
  - Usar como claves en diccionarios
  - Representar registros fijos

Colecciones

Sets

- **Creación:** nombre_conjunto = {valor 1, valor 2, ...}
- **Acceso:** no se accede por posición
- **Modificación:** no se cambia por índice
- **Ordenada:** no mantienen orden ni índices

- **Usos**
  - Eliminar valores repetidos automáticamente
  - Operaciones de conjuntos
  - Verificar pertenencia de manera rápida

Colecciones

Diccionarios

Estructura que **almacena datos en pares clave–valor**, donde cada clave es única y se asocia a un valor.

- **Creación**
  - nombre_dict = {"clave": "valor"}
  - nombre_dict = {}
- **Acceso**
  - nombre_dict["clave"]
  - nombre_dict.get("clave", valor_defecto)
- **Modificación:** nombre_dict["clave"] = nuevo_valor
- **Ordenada:** mantienen el orden de inserción
- **Usos**
  - Buscar valores rápidamente a partir de una clave
  - Representar datos estructurados
  - Modelar entidades con atributos

Colecciones

Diccionarios - métodos

Método	Descripción	Ejemplo
keys()	Devuelve una vista con todas las claves	persona.keys()
values()	Devuelve una vista con todos los valores	persona.values()
items()	Devuelve pares (clave, valor) como tuplas	persona.items()
get(clave, [def])	Devuelve el valor de la clave o el valor por defecto	persona.get("edad", 0)
update(otro_dic)	Actualiza con pares de otro diccionario	persona.update({"edad": 30})
pop(clave, [def])	Elimina una clave y devuelve su valor	persona.pop("edad", None)
popitem()	Elimina y devuelve el último par clave-valor agregado	persona.popitem()
setdefault(clave, [def])	Devuelve el valor de la clave; si no existe, la crea con el valor por defecto	persona.setdefault("ciudad", "Córdoba")
clear()	Elimina todos los elementos del diccionario	persona.clear()
copy()	Devuelve una copia superficial del diccionario	nuevo = persona.copy()

Lista de compras

Crea un programa interactivo que registre productos y calcule información de la compra.

1. Pedir al usuario el nombre y precio de 3 productos
2. Guardar los datos en una estructura que permita acceder tanto al nombre como al precio
3. Calcular el total a pagar y determinar el producto más caro
4. Mostrar la lista de productos, el total y cuál fue el producto de mayor costo

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**

Actividades iniciales

Trabajo en equipo

1. Crea una **carpeta en tu PC** “Tu nombre - Proyecto Aurelion”
2. **Conecta** dicha carpeta a VS Code
3. Dentro, crea el archivo **Documentación.md**
4. Completa el **tema**, **problema** y **solución**
5. Describe **estructura**, **tipos** y **escala** de la BD

Para el programa en Python que consultará la documentación:

1. Define **información**, **pasos** y **pseudocódigo**

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**