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

El **editor de texto** es una herramienta que permite crear, modificar o eliminar texto, muchos editores incorporan funciones más especializadas para la **programación**, por esta razón se les suele denominar como **editores de código**.

Un editor de texto puede ser tan básico como un block de notas de tu computadora, o algo más profesional para desarrollo **Visual Studio Code**.

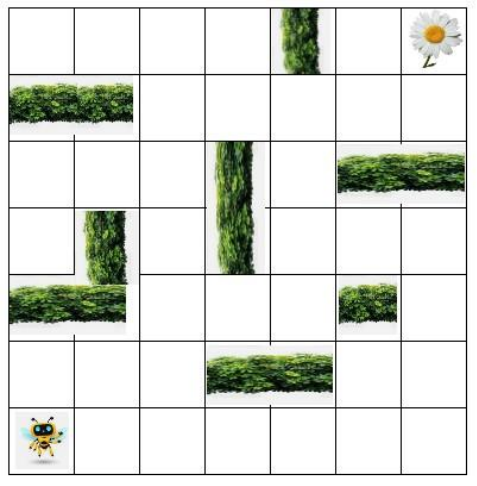
Un editor de texto también nos ayuda a la hora de hacer codigo:
subraya, muestra errores y generalmente tienen la función de sintaxis high lighter, lo que significa que asigna colores según cada tipo de elemento en el código, para poder leerlo más fácil.

https://code.visualstudio.com/download

¿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 la Programación?



El objetivo del robot abeja es llegar hasta la flor, esquivando los arbustos que hay de por medio. Para esto, hay que decirle cómo se debe mover, pero solo puede saltar de a un casillero por vez. La forma en que se le puede "decir" es a través de instrucciones y estas son: "arriba", "abajo", "izquierda" y "derecha". Por supuesto, teniendo en cuenta que debe moverse dentro del cuadrilátero.

- ¿Cuántas instrucciones son necesarias?
- Al ser más de una instrucción ¿Es importante el orden en que se ejecutan?
- ¿Hay más de una forma en que el robot abeja pueda llegar a la flor?
- ¿Hay un camino óptimo?
- ¿Cómo puedo medir si un camino es el óptimo con respecto a otros caminos?

Cualquiera de las siguientes, podrían ser una solución:

- Derecha, Derecha, Arriba, Arriba, Derecha, Derecha, Arriba, Arriba, Arriba, Derecha, Derecha, Arriba

- Derecha, Derecha, Derecha, Derecha, Derecha, Derecha, Arriba, Arriba, Arriba, Izquierda, Izquierda, Arriba, Arriba, Derecha, Arriba, Derecha

- Arriba, Derecha, Derecha, Arriba, Arriba, Arriba, Arriba, Arriba, Derecha, Derecha, Derecha, Derecha, Arriba

Lo importante, en principio, es que notemos que al resolver el problema planteado, lo que estamos haciendo, utilizando un pensamiento lógico, es precisamente, programación

Entonces, **programar, es armar una secuencia lógica de pasos a seguir, en pos de cumplir un objetivo.**

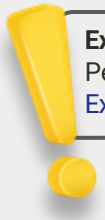
Preparar entorno

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 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

No existe *una lista de "todos" los comandos de Python, sino funciones, sentencias, palabras clave y herramientas de línea de comando que se utilizan para programar.*

Aquí los mas usados...

Funciones integradas comunes

- `print()`: Muestra mensajes o valores en la consola.
- `input()`: Permite obtener datos del usuario desde la entrada estándar.
- `len()`: Devuelve la longitud (el número de elementos) de un objeto, como una cadena de texto o una lista.
- `type()`: Indica el tipo de dato de una variable.
- `range()`: Genera una secuencia de números, útil en bucle

Sentencias y estructuras de control

- `if` / `else` / `elif`: Para tomar decisiones y ejecutar código condicionalmente
- `for` / `while`: Bucles para repetir un bloque de código.
- `def`: Para definir funciones personalizadas.

Palabras clave y tipos de datos

- **Variables**:

Para almacenar diferentes tipos de datos, como números enteros (`int`), decimales (`float`), cadenas de texto (`str`), y valores booleanos (`bool`).

- **Estructuras de datos**:

Como listas, diccionarios y conjuntos, que permiten organizar y almacenar colecciones de datos.

Herramientas de línea de comando (para gestionar Python)

- `pip`: Herramienta para instalar y gestionar paquetes de software de Python.
- `python`: El comando principal para ejecutar scripts de Python y entrar en el intérprete interactivo.

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

El Zen de Python

Para conocer mejor el lenguaje que estaremos aprendiendo existe una colección de los 19 principios que influyen en el diseño del lenguaje Python.

De alguna manera, muestran la filosofía del mismo:

1.Bello es mejor que feo.

2.Explícito es mejor que implícito.

3.Simple es mejor que complejo.

4.Complejo es mejor que complicado.

5.Plano es mejor que anidado.

6.Espaciado es mejor que denso.

7.La legibilidad es importante.

8.Los casos especiales no son lo suficientemente especiales como para romper las reglas.

9.Sin embargo la practicidad gana a la pureza.

10.Los errores nunca deben pasar silenciosamente.

11.A menos que se silencien explícitamente.

12.**Frente a la ambigüedad, evitar la tentación de adivinar.**

13.**Debería haber una, y preferiblemente solo una, manera obvia de hacerlo**.

14.A pesar de que esa manera no sea obvia a menos que seas Holandés (el creador lo era)

15.Ahora es mejor que nunca.

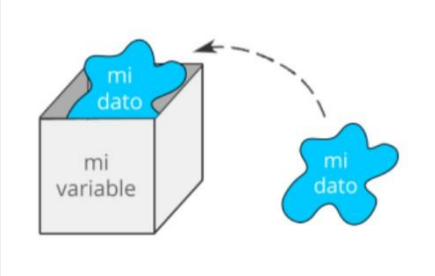
16.A pesar de que nunca es muchas veces mejor que ahora mismo.

17.**Si la implementación es difícil de explicar, es una mala idea.**

18.Si la implementación es fácil de explicar, puede que sea una buena idea.

Variables

Una variable es un espacio de memoria donde guardamos un dato, ese espacio de memoria a la vez recibe un nombre y esto conforma la estructura de datos más simple que podemos encontrar.



Contenedores que **almacenan valores reutilizables**.

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

Reglas ,nomenclatura

•El nombre no puede empezar con un número
Por ejemplo "mi_variable" sería correcto, pero "123mi_variable" sería erróneo

•No se permite el uso de guiones del medio -
Por ejemplo "mi-variable" sería erróneo

•No se permite el uso de espacios.
Por ejemplo "mi variable" sería erróneo

•No usar nombres reservados para Python. Las palabras reservadas son utilizadas por Python internamente, por lo que no podemos usarlas para nuestras variables o funciones.
Por ejemplo, a veces podríamos usar "int" ó "for" y esto nos daría error, porque como se verá más adelante, esas palabras son parte de la sintaxis de Python.

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

Tipos de Datos

•Enteros: el conjunto de números enteros

•Floats: el conjunto de números reales o de punto flotante

•Cadenas o Strings: es texto, caracteres alfanuméricos que se introducen entre comillas dobles o simples

•Booleanos: representan Verdadero ó Falso

•Complejos: el conjunto de números complejos

Todo valor que pueda ser asignado a una variable tiene asociado un tipo de dato y esto establece qué operaciones se pueden realizar sobre la misma.

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

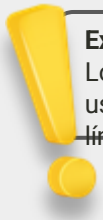
Elementos que realizan cálculos, comparaciones y evaluaciones en el código.

- **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

Algunos ejemplos en Python:

```
>>> a = 'Hola '
>>> b = 'Mundo !'
>>> print(a + b)
Hola Mundo !

>>> x = 3
>>> y = 12
>>> print(x + y)
15

>>> print(a + x)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_18232/136165486.py in <module>
----> 1 print(a + x)

TypeError: can only concatenate str (not "int") to str
```

Algunos ejemplos en Python:

```
>>> # Dividir "y" entre "x"
>>> y = 9
>>> x = 3
>>> print(y/x)
3.0

>>> # Potencia de "y" elevado a la "x"
>>> y = 2
>>> x = 4
>>> print(y**x)
16

>>> # Devolver el resto de la división
>>> y = 13
>>> x = 3
>>> print(y%x)
1

>>> # Imprimir el tipo de dato de una variable y una constante
>>> y = 2.5
>>> type(y)
float
>>> type(2.5)
float
```

¿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()

Lista

Una estructura de datos muy importante en Python es la lista, que consiste en una serie de elementos ordenados. Esos elementos pueden ser de distinto tipo, e incluso pueden ser de tipo lista también.

Operaciones con listas:

- Creacion

```
mi_lista = ['Rojo','Azul','Amarillo','Naranja','Violeta','Verde']
```

- Imprimir

```
print(mi_lista)
```

- Ver el tipo de dato

```
type(mi_lista)
```

Lista

- Imprimir el tercer elemento de la lista (el índice comienza en cero)

```
print(mi_lista[2])
```

- Acceder a un rango dentro de la lista (el límite inferior se incluye y el superior se excluye)

```
print(mi_lista[0:2])
```

- Al no poner primer valor, Python asume que es un 0

```
>>> print(mi_lista[:2])
['Rojo', 'Azul']
```

- Al no poner segundo valor, Python asume que se trata de todos los elementos a partir del primero

```
 >>> print(mi_lista[0:])
 ['Rojo', 'Azul', 'Amarillo', 'Naranja', 'Violeta', 'Verde']
```

- Agregar un elemento al final de la lista (Si el elemento ya existe va a quedar duplicado)

```
mi_lista.append('Blanco')
```

Lista

- Extraer y recuperar el último elemento de la lista

```
>>> ultimo = mi_lista.pop()
>>> print(ultimo)
Gris
```

- Multiplicar la lista 3 veces

```
>>> print(['a','b','c'] * 3)
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

- Ordenar una lista de menor a mayor

```
>>> lista= [1,4,3,6,8,2]
>>> lista.sort()
>>> print(lista)
[1,2,3,4,6,8]
```

- Ordenar una lista de mayor a menor

```
>>> lista= [1,4,3,6,8,2]
>>> lista.sort(reverse=True)
>>> print(lista)
[8,6,4,3,2,1]
```

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

Tuplas

Son una estructura similar a las listas, la diferencia está en que no se pueden modificar una vez creadas, es decir que son **inmutables**:

- Convertir una lista a tupla

```
mi_tupla=tuple(mi_lista)
```

- Imprimir el índice 1 de la tupla

```
>>> print(mi_tupla[1])
Azul
```

- Evaluar si un elemento está contenido en la tupla (Devuelve un valor booleano)

```
>>> 'Rojo' in mi_tupla
True
```

- Evaluar las veces que está un elemento específico

```
>>> mi_tupla.count('Rojo')
1
```

- Tupla con un solo elemento

```
mi_tupla_unitaria = ('Blanco',)
```

- Empaquetado de tupla, tupla sin paréntesis

```
mi_tupla='Gaspar', 5, 8, 1999
```

Tuplas

- Desempaquetado de tupla, se guardan los valores en orden de las variables

```
>>> nombre, dia, mes, año = mi_tupla
>>> print("Nombre: ", nombre, " - Dia:", dia, " - Mes: ", mes, " - Año: ", año)
Nombre:  Gaspar  - Dia: 5  - Mes:  8  - Año:  1999
```

- Convertir una tupla en una lista

```
mi_lista=list(mi_tupla)
```

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()

Diccionario

Un diccionario tiene una organización de 'clave' y 'valor':

- Crear un diccionario

```
mi_diccionario = {  'Colores Primarios': ['Rojo','Azul','Amarillo'],
                    'Colores secundarios': ['Naranja','Violeta','Verde'],
                    'Clave3': 10,
                    'Clave4': False}
```

- Imprimir un valor a través de su clave

```
>>> print(mi_diccionario['Colores secundarios'])
['Naranja', 'Violeta', 'Verde']
```

- Agregar un valor

```
mi_diccionario['Clave5']='Otro ejemplo'
```

- Cambiar un valor

```
mi_diccionario['Clave3']=2
```

Diccionarios

- Cambiar un valor

```
mi_diccionario['Clave3']=2
```

- Eliminar un elemento de un diccionario a través de su clave

```
del mi_diccionario['Clave4']
```

- Utilizar una tupla como clave de un diccionario

```
mi_tupla=("Argentina", "Italia", "Inglaterra")
mi_diccionario={mi_tupla[0]:"Buenos Aires",
                mi_tupla[1]:"Roma",
                mi_tupla[2]:"Londres"}
```

- Colocar una tupla dentro de un diccionario

```
mi_diccionario={'Clave1':'Valor1', 'Clave2':(1,2,3,4,5)}
```

- Colocar una lista dentro de un diccionario (Notar que la diferencia está en el paréntesis '()' y el corchete '[]')

```
 mi_diccionario={'Clave1':'Valor1', 'Clave2':[1,2,3,4,5]}
```

- Colocar un diccionario dentro de un diccionario

```
mi_diccionario={'Clave1':'Valor1', 'Clave2':{'numeros':[1,2,3,4,5]}}
```

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

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