IPN-CICATA
Querétaro

24 de junio de 2023

Python 101

M.T.A Angel Moisés Hernández Ponce


INTERNATIONAL CONFERENCE ON APPLIED SCIENCE AND ADVANCED TECHNOLOGY
ICASAT 2023
INTERNATIONAL CONFERENCE ON APPLIED SCIENCE AND ADVANCED TECHNOLOGY
Conference - Workshops - Posters - Querétaro, México.

Qué es Python?

- Python es un lenguaje de programación interpretado, de alto nivel, de tipado dinámico y orientado a objetos con una filosofía amigable.
- Python es un lenguaje de código abierto.
- Corre en cualquier sistema operativo.

Quién usa Python?

Para qué se usa Python?

Python se puede utilizar para una variedad de aplicaciones, las más importantes actualmente son:

- Inteligencia Artificial
- Computo científico
- Ciencia y Análisis de datos
- Desarrollo de interfaces gráficas de usuario
- Frameworks para páginas web
- Procesamiento de imágenes
- Desarrollo de videojuegos

La filosofía de Python

Bello es mejor que feo
Explicito es mejor que implicito
Simple es mejor que complejo
Complejo es mejor que complicado[...]
Si la idea es facil de explicar es una buena idea
Si la idea es dificil de explicar es una mala idea

Cómo usar Python?

IDE	Entorno Virtual	Entorno Virtual Online
• Tiene una interfaz integrada para escribir, editar, compilar y depurar código • Tiene herramientas adicionales para analizar errores, realizar pruebas y administra proyectos. • Interfaz gráfica cómoda para programar.	• Entornos aislados para proyectos independientes. • Permite organizar y mantener limpios los proyectos de desarrollo. • Facilita la colaboración entre varios usuarios.	• Combina un IDE y un entorno virtual pero con la ventaja de que corre en la nube, es decir, no necesitamos instalar software en nuestra computadora. • Viene con muchos paquetes y librerías precargados. • Integración con otros servicios en la nube

Google Colaboratory

https://colab.research.google.com

Configuración del proyecto

Login

Te damos la bienvenida a Colaboratory

Nombre del proyecto

Botón de compartir

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir

Acceder

Barra de herramientas

Índice

+ Código + Texto Copiar en Drive

Conectar

Introducción

Ciencia de datos

Aprendizaje automático

Más recursos

Ejemplos destacados

Sección

Gestor de celdas

índice del proyecto

Te damos la bienvenida a Colab

Si ya conoces Colab, mira este video para aprender sobre las tablas interactivas, la vista histórica de código ejecutado y la paleta de comandos.

¿Qué es Colab?

Colab, o "Colaboratory", te permite escribir y ejecutar código de Python en tu navegador, con

- Sin configuración requerida
- Acceso sin costo a GPU
- Facilidad para compartir

Celdas de texto

Seas **estudiante**, **científico de datos** o **investigador de IA**, Colab facilita tu trabajo. Mira este video introductorio sobre Colab para obtener más información, o bien comienza a usarlo más abajo.

Introducción

El documento que estás leyendo no es una página web estática, sino un entorno interactivo denominado **notebook de Colab**, que permite escribir y ejecutar código.

Por ejemplo, esta es una **celda de código** con una secuencia de comandos Python corta que calcula un valor, lo almacena en una variable y devuelve el resultado:

```
[ ]  seconds_in_a_day = 24 * 60 * 60
     seconds_in_a_day
```

Celda de código

Herramientas adicionales

Fundamentos de progamación en Python

Tipos de variables

Qué es una variable?

- Una variable se utiliza para almacenar información o datos para que podamos llamarlas cuando lo necesitemos.
- Una variable siempre debe tener un nombre, de preferencia que haga referencia a lo que contiene.
- A diferencia de otros lenguajes de programación, en Python no necesitamos declarar el tipo de variable.
- También podemos cambiar el contenido de la variable sobre la marcha, incluso cambiar el tipo de variable

Algunas restricciones

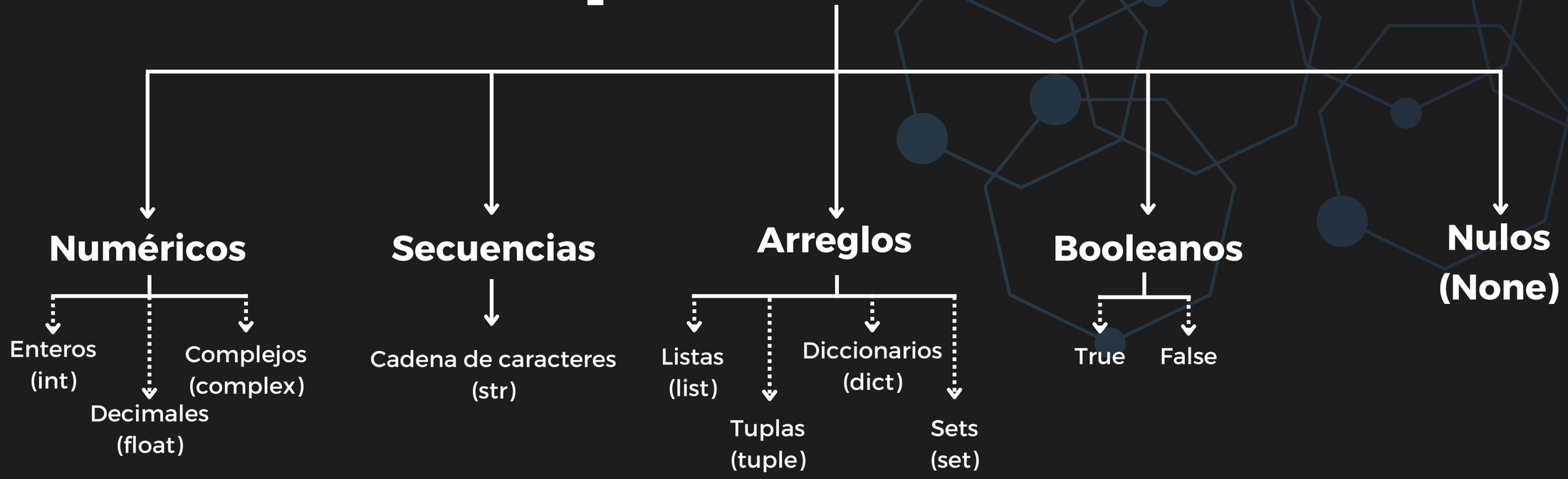
- No pueden iniciar con números
  - 2x, 20kilos, 345i
- No puede haber espacios en el nombre
- No puede contener estos simbolos ' " <> / \ ? ! ( ) @ $%&*+-
- No pueden utilizar palabras clave como: for, while, if, else, help, or, and

Palabras reservadas

and	**as**	assert	break	class
continue	def	del	elif	else
except	False	finally	for	from
global	if	import	in	is
lambda	none	nonlocal	not	or
pass	raise	return	True	try
while	with	yield		

Tipos de datos

Tipos de datos



Pro tip: Si no están seguros del tipo de dato de una variable, pueden escribir type(nombre_variable) y Python les dirá el tipo.

Numéricos

- Puede ser cualquier número.
- Los enteros se conocen como int
- Los números con decimales se llaman flotantes
- Se puede realizar cualquier operación aritmética con ellos, comparaciones lógicas.
- Los números complejos deben tener su parte real y su parte imaginaria.

Operadores numéricos

Nombre	Simbolo	Ejemplo
Suma	+	1 + 1
Resta	-	5 - 1
Multiplicación	*	10*4
División	/	20/4
Modulo	%	24%3
Potencia	**	2**3
División por suelo (floor division)	//	15//2

Jerarquía de operaciones

Exponenciación/Potenciación

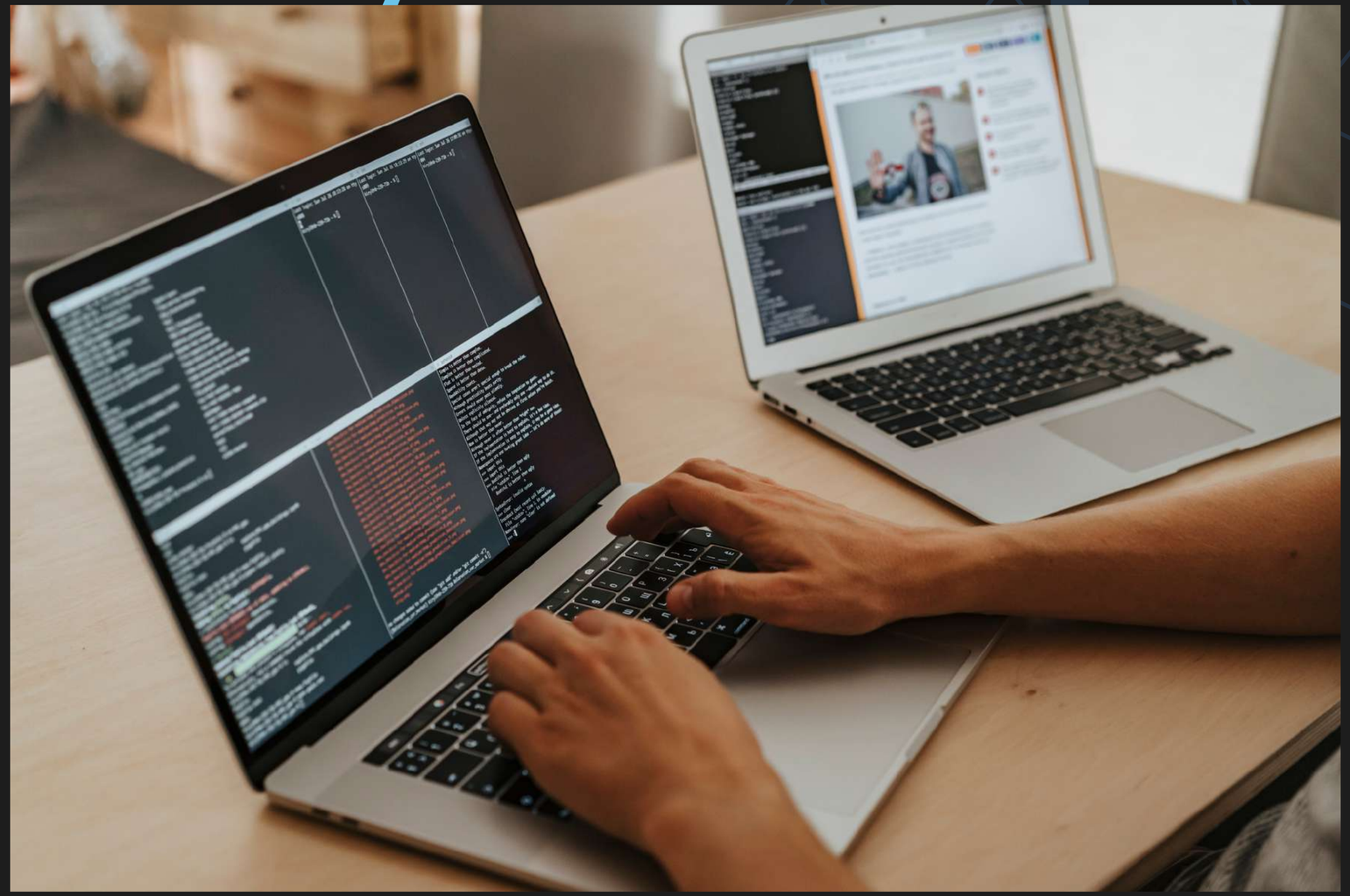
Negación

Multiplicación / División
División por Suelo/ Módulo

Suma y Resta

Los paréntesis también influyen en la jerarquía de las operaciones!

Practica 1 — Python como calculadora



Cadenas de caracteres (strings)

- Pueden ser cualquier carácter siempre y cuando estén rodeados por comillas simples (') o comillas dobles ("").
  - Letras (mayúsculas y minúsculas) y dígitos numéricos (0-9)
  - Espacios en blanco
  - Caracteres especiales
  - Caracteres de escape
    - \n --> Salto de línea
    - \t --> Tabulación
    - \a --> Produce un beep (solo funciona en la terminal)
  - Unicode (casi cualquier lenguaje humano es soportado)

Strings pt2

- En la mayoría de los casos, las strings son de linea simple, es decir, no hay un 'enter' entre ellas.
- Las strings multilineas se ponen entre tres comillas dobles (""" """)
- Si nuestra string tiene un apostrofe (´) es mejor usar comillas dobles (" ")
- Con los acentos no hay problema.

Practica 2 — Strings

Manipulación de strings

- Las strings se pueden manipular ya que es una cadena de caracteres.
- La operación de indexación (indexing) permite extraer un carácter específico.
  - Se utilizan los corchetes []
- La operación de slicing extrae una parte del string
  - Se utilizan los corchetes [] pero indicando el rango deseado.

En Python la cuenta inicia desde 0
Los espacios también cuentan

Ejemplo

```python
# Operación de indexación y slicing de strings
texto = "Python es genial"

# Indexación
print(texto[0])     # Acceder al primer carácter, imprime: "P"
print(texto[7])     # Acceder al octavo carácter, imprime: "e"
print(texto[-1])    # Acceder al último carácter, imprime: "l"

# Slicing
print(texto[0:6])   # Obtener los caracteres desde el índice 0 hasta el 5, imprime: "Python"
print(texto[7:])    # Obtener los caracteres desde el índice 7 hasta el final, imprime: "es genial"
print(texto[:6])    # Obtener los caracteres desde el inicio hasta el índice 5, imprime: "Python"
print(texto[-6:])   # Obtener los últimos 6 caracteres, imprime: "genial"
print(texto[::3])   # Obtener cada segundo carácter, imprime: "Ph nl"
```

Ejemplo 2

```python
# Operación de indexación y slicing de strings
texto = "Hola, bienvenidos al curso de Python 101"

# Indexación
print(texto[0])       # Acceder al primer carácter, imprime: "H"
print(texto[5])       # Acceder al sexto carácter, imprime: ","
print(texto[-1])      # Acceder al último carácter, imprime: "1"

# Slicing
print(texto[0:4])     # Obtener los caracteres desde el índice 0 hasta el 3, imprime: "Hola"
print(texto[7:])      # Obtener los caracteres desde el índice 7 hasta el final, imprime: "bienvenidos al curso de Python 101"
print(texto[:4])      # Obtener los caracteres desde el inicio hasta el índice 3, imprime: "Hola"
print(texto[-3:])     # Obtener los últimos 3 caracteres, imprime: "101"
print(texto[::2])     # Obtener cada segundo carácter, imprime: "Hl,bevnoslcus ePto 1"
```

Practica 3 — Manipulación de strings

Arreglos

Listas

- Arreglos no ordenados
- Pueden contener cualquier tipo de dato, incluso otra lista.
- Son mutables
- Se pueden añadir o eliminar valores según se necesite.
- Se declaran con corchetes []
- Cada valor se separa con una coma (,)

Ejemplo

```python
numeros = [1, 2, 3, 4, 5]  # Lista de números
nombres = ["Juan", "María", "Carlos"]  # Lista de strings
mezclado = [1, "dos", True, [1, 2, 3]]  # Lista con elementos de diferentes tipos

print(numeros)  # Imprime: [1, 2, 3, 4, 5]
print(nombres)  # Imprime: ["Juan", "María", "Carlos"]
print(mezclado)  # Imprime: [1, "dos", True, [1, 2, 3]]
```

Manipulación de listas

Nombre	Función	Sintaxis
append()	Añade un elemento al final de la lista	append(elemento)
insert()	Inserta un elemento en un índice específico de la lista	insert(índice, elemento)
remove()	Elimina la primera aparición de un elemento	remove(elemento)
pop()	Elimina y devuelve el elemento en el índice especificado. Si no se proporciona el índice, se elimina y devuelve el último elemento	pop([indice])
index()	Devuelve el índice de la primera aparición de un elemento en la lista	index(elemento)
count()	Cuenta el número de veces que un elemento aparece en la lista	count(elemento)
sort()	Ordena los elementos de la lista de forma ascendente	nombre_lista.sort()
reverse()	Invierte el orden de los elementos en la lista	nombre_lista.reverse()
clear()	Elimina todos los elementos de la lista	nombre_lista.clear()

Ejemplo métodos

```python
# append(elemento)
frutas = ['manzana', 'pera', 'banana']
frutas.append('naranja')
print(frutas)  # Imprime: ['manzana', 'pera', 'banana', 'naranja']
```

```python
# insert(indice, elemento)
colores = ['rojo', 'verde', 'azul']
colores.insert(1, 'amarillo')
print(colores)  # Imprime: ['rojo', 'amarillo', 'verde', 'azul']
```

```python
# remove(elemento)
animales = ['perro', 'gato', 'perro', 'conejo']
animales.remove('perro')
print(animales)  # Imprime: ['gato', 'perro', 'conejo']
```

```python
# pop([indice])
numeros = [1, 2, 3, 4, 5]
elemento = numeros.pop(2)
print(numeros)  # Imprime: [1, 2, 4, 5]
print(elemento)  # Imprime: 3
```

```python
# index(elemento)
frutas = ['manzana', 'pera', 'banana']
indice = frutas.index('pera')
print(indice)  # Imprime: 1
```

```python
# count(elemento)
numeros = [1, 2, 2, 3, 2, 4]
contador = numeros.count(2)
print(contador)  # Imprime: 3
```

```python
# sort()
numeros = [4, 2, 1, 3]
numeros.sort()
print(numeros)  # Imprime: [1, 2, 3, 4]
```

```python
# reverse()
letras = ['a', 'b', 'c', 'd']
letras.reverse()
print(letras)  # Imprime: ['d', 'c', 'b', 'a']
```

```python
# clear()
elementos = [1, 2, 3, 4, 5]
elementos.clear()
print(elementos)  # Imprime: []
```

Practica 4 — Listas y manipulación

Tuplas

- Las tuplas son un tipo de arreglos inmutable, es decir, no se pueden modificar una vez son declaradas.
- Es una colección ordenada.
- Se declaran con paréntesis ().
- Son útiles para guardar valores constantes o que nunca van a cambiar.
- Se pueden aplicar la indexación y slicing a las tuplas.
- Algunos métodos están restringidos por la naturaleza de las tuplas.

Ejemplos de tuplas

```python
# Tupla de nombres de colores
colores = ('rojo', 'verde', 'azul')

# Tupla de coordenadas en un plano
coordenadas = (3.5, -2.8)

# Tupla de información personal
informacion_personal = ('Juan', 'Pérez', 25, 'Calle Principal')

# Tupla de días de la semana
dias_semana = ('Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo')

# Tupla de coordenadas geográficas
coordenadas_geograficas = (40.7128, -74.0060)
```

Practica 5 — Tuplas

Diccionarios

- Son arreglos no ordenados.
- Relacionan los datos la forma llave-valor.
- Las llaves pueden deben ser valores inmutables como strings, números o tuplas.
- Los valores asociados a cada llave puede ser cualquier tipo de dato, incluyendo otros diccionarios.
- Se declaran con las llaves {}.
- Después de escribir la llave se escriben dos puntos(:) para indicar los valores asociados a esa llave, cada par llave-valor se separa por una coma.

Ejemplos diccionarios

```python
persona = {
    'nombre': 'Juan',
    'edad': 25,
    'ciudad': 'Madrid'
}

print(persona['nombre'])  # Imprime: 'Juan'
print(persona['edad'])    # Imprime: 25
print(persona['ciudad'])  # Imprime: 'Madrid'


persona['edad'] = 26   # Modificar el valor de 'edad'
persona['genero'] = 'Masculino'   # Agregar una nueva pareja clave-valor

print(persona)  # Imprime: {'nombre': 'Juan', 'edad': 26, 'ciudad': 'Madrid', 'genero': 'Masculino'}
```

Ejemplo Diccionarios

```python
generos_music = {
    'rock': {
        'origen': 'Estados Unidos',
        'decada': 'década de 1950',
        'artistas_destacados': ['The Beatles', 'Led Zeppelin', 'Queen'],
        'canciones_famosas': ['Stairway to Heaven', 'Bohemian Rhapsody']
    },
    'jazz': {
        'origen': 'Estados Unidos',
        'decada': 'finales del siglo XIX',
        'artistas_destacados': ['Louis Armstrong', 'Miles Davis', 'Ella Fitzgerald'],
        'canciones_famosas': ['Take Five', 'Summertime']
    },
    'pop': {
        'origen': 'Estados Unidos',
        'decada': 'década de 1950',
        'artistas_destacados': ['Michael Jackson', 'Madonna', 'Taylor Swift'],
        'canciones_famosas': ['Thriller', 'Like a Prayer']
    }
}


print(generos_music['rock']['origen'])  # Imprime: 'Estados Unidos'
print(generos_music['jazz']['artistas_destacados'])  # Imprime: ['Louis Armstrong', 'Miles Davis', 'Ella Fitzgerald']
print(generos_music['pop']['canciones_famosas'][0])  # Imprime: 'Thriller'
```

Practica Diccionarios

Sets

- Los sets es un arreglo no ordenado que almacena datos únicos, es decir, no admite duplicados.
- Son útiles cuando necesitamos almacenar datos sin orden específico y únicos.
- Se declaran usando las llaves {} y los datos se separan con coma (,).
- Los datos no están indexados por lo que no podemos hacer uso de la indexación o slicing en los datos, sin embargo, sí podemos interactuar con los sets de otras formas.
- Los sets son mutables

Ejemplo Sets

```python
# Sets
numeros = {1, 2, 3, 4, 4, 5}
print(numeros)  # Imprime: {1, 2, 3, 4, 5}

# Un set con varios tipos de datos
datos = {1, "Hola", True, 3.14}
print(datos)  # Imprime: {1, 'Hola', True, 3.14}

# También pueden declarar un conjunto vacío
conjunto_vacio = set()
print(conjunto_vacio)  # Imprime: set()

# Modificando valores de los sets
frutas = {"manzana", "plátano", "naranja"}

print("manzana" in frutas)  # Verifica si "manzana" está en el conjunto

frutas.add("mango")  # Agrega un elemento al conjunto

frutas.remove("plátano")  # Elimina un elemento del conjunto

print(len(frutas))  # Devuelve la cantidad de elementos en el conjunto
```

Ejemplo de métodos con Sets

```python
# Aplicando varios métodos a los sets
# Creación de un conjunto
frutas = {"manzana", "plátano", "naranja"}

# Método add()
frutas.add("mango")
print(frutas)  # Imprime: {'manzana', 'plátano', 'naranja', 'mango'}

# Método remove()
frutas.remove("plátano")
print(frutas)  # Imprime: {'manzana', 'naranja', 'mango'}

# Método pop()
elemento = frutas.pop()
print(elemento)  # Imprime un elemento aleatorio del conjunto
print(frutas)   # Imprime el conjunto sin el elemento removido

# Método clear()
frutas.clear()
print(frutas)  # Imprime: set()
```

Mas ejemplos de métodos

```python
# Creación de dos conjuntos para ilustrar otros métodos
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Método union()
union = set1.union(set2)
print(union)  # Imprime: {1, 2, 3, 4, 5, 6, 7, 8}

# Método intersection()
interseccion = set1.intersection(set2)
print(interseccion)  # Imprime: {4, 5}

# Método difference()
diferencia = set1.difference(set2)
print(diferencia)  # Imprime: {1, 2, 3}
```

Operadores booleanos & de comparación

Operadores booleanos

- Sirven para establecer comparaciones lógicas entre dos valores booleanos, True (Verdadero) y False (False).
- Están basados en el algebra booleano desarrollada por George Boole
- El resultado siempre sera de tipo booleano (bool).
- Solo hay dos resultados posibles, True o False.
- Las tablas de verdad nos indican los resultados posibles de las combinaciones.

Tablas de verdad

Compuerta AND

Entrada		Salida
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Compuerta OR

Entrada		Salida
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

Pro tip: En la compuerta AND la salida, siempre es 1 si ambas entradas son 1. En la compuerta OR, la salida siempre es 1 a menos que ambas entradas sean 0.

Operadores de comparación

- Los operadores de comparación evalúan si una afirmación es verdadera.
- La comparación debe ser entre un mismo tipo de valores.
- El resultado siempre es de tipo bool.
- Nos sirven para controlar el flujo de un programa o determinar cierto tipo de acciones según sea el caso.

Operador	Afirmacion
==	Es igual a
!=	Es distinto a
<	Menor que
>	Mayor que
<=	Menor o igual que
>	Mayor o igual que

Ejemplos Operadores

```python
# Operadores booleanos
a = True
b = False

print(a and b)  # Imprime: False
print(a or b)   # Imprime: True
print(not a)    # Imprime: False

# Otra forma de usar los operadores booleanos
print(a & b) # Imprime False
print(a | b) # Imprimie True

# Operadores de comparación
x = 5
y = 10

print(x == y)  # Imprime: False
print(x != y)  # Imprimie: True
print(x < y)   # Imprime: True
print(x >= y)  # Imprime: False
```

Practica Operadores

Examen sorpresa

Estructuras de control

Estructuras de control

Sintaxis

Palabra clave

Condición a evaluar

Estos : (dos puntos) son obligatorios

```
>>> if var < 200:
...     print('Afirmativo')
```

Este pequeño espacio se llama identación

Instrucciones a ejecutar dentro de la condición

```
Afirmativo
```

Resultado

if-else-elif

- En las condicionales, la condición que está hasta arriba se evalúa primero y así sucesivamente hacia abajo.
- Pueden hacer múltiples evaluaciones con la palabra "elif" y siempre deben escribir la condición a evaluar.
- La palabra 'else' siempre indica la acción a realizar cuando no se cumpla la condición.

```python
num = int(input("Ingresa un número: "))

if num > 0:
    print("El número es positivo")
elif num < 0:
    print("El número es negativo")
else:
    print("El número es cero")
```

Ejemplos

```python
if calificacion >= 90:
    print("Aprobado con A")
elif calificacion >= 80:
    print("Aprobado con B")
elif calificacion >= 70:
    print("Aprobado con C")
elif calificacion >= 60:
    print("Aprobado con D")
else:
    print("Reprobado")
```

Ejemplos

```
if num % 2 == 0:
    print("El número es par")
else:
    print("El número es impar")
```

Practica If-Else

Estructuras repetitivas

Ciclo for

- Se utiliza para iterar sobre una secuencia o arreglo de datos.
- A diferencia del ciclo while, este ciclo itera sobre todos los elementos del arreglo.
- Se pueden ejecutar acciones mientras el ciclo esté activo.
- Si van a iterar sobre un rango de valores se usa la palabra clave range().

Ciclo while

- Este ciclo se ejecuta siempre y cuando se cumpla una condición.
- Si la condición no se cumple o está mal planteada, tendremos un bucle infinito y NO queremos eso.
- Se usa una variable contadora para evitar los ciclos infinitos.
- Se puede interrumpir de manera arbitraria con la palabra clave break.

Sintaxis

Ciclo for

```
for elemento in secuencia:
    # Código a ejecutar para cada elemento
```

Ciclo while

```
while condicion:
    # Código a ejecutar mientras se cumpla la condición
```

Ejemplo ciclo for

```python
for fruta in frutas:  # Itera sobre cada elemento de la lista 'frutas'
    print(fruta)  # Imprime el valor de cada elemento


# Ahora usamos animales en lugar de frutas
animales = ["león", "tigre", "jirafa", "elefante"]

for animal in animales:
    print(animal) # Imprime cada animal de la lista


# Usando la función range()
for num in range(1, 6):
    print(num) # Imprime una lista de números del 0 al 5
```

Ejemplo ciclo while

```python
# Ejemplo de ciclos while

contador = 0 # Variable contadora

while contador < 5:
    print(contador) # Imprime el valor de la variable contador
    contador += 1 # Incrementa en 1 el valor de contador


# Usando ciclo while y la estructura if
respuesta = "" # String vacía

while respuesta != "si":
    respuesta = input("¿Quieres continuar? (si/no): ")

    if respuesta == 'si':
        print('Cotinuemos')
    else:
        print('Ciclo cancelado')

# Ejemplo 3
suma = 0
numero = 1

while numero != 0:
    numero = int(input("Ingresa un número (0 para salir): "))
    suma += numero

print("La suma total es:", suma)
```

Practica Ciclos

Funciones

Funciones personalizadas

- Las funciones personalizadas nos permite encapsular código estructurado y reutilizarlo cuando sea necesario.
- Podemos definir tantas funciones como queramos y utilizarlas para tareas específicas.
- Pueden tener cualquier tipo de variable como entrada, incluso puede no tener entradas.
- Para definir una función se usa la palabra clave **def**

Sintaxis

Palabra clave **def**

No olviden los dos puntos

```
def nombre_de_la_funcion(parametros):
    # Código de la función
    # ...
    return resultado
```

Parámetros de entrada

Palabra clave **return**

Ejemplo 1 Funciones

```
def sumar(a, b):
    resultado = a + b
    return resultado
```

Todas las variables que declaren dentro de la función son temporales, es decir, existen solamente mientras la función se ejecute. Si quiere llamar a la variable fuera de la función les arrojará un error

Ejemplo 2 Funciones

```
def es_positivo(numero):
    if numero > 0:
        return True
    else:
        return False
```

Más sobre los parámetros

- Pueden asignar un valor por default, en caso de ser necesario.
- El número de parámetros de la función siempre tiene que ser igual los valores de entrada. De lo contrario les marcara un error.
- Cada parámetro lo deben usar en la función.
- Recuerden que cada parámetro debe estar separado por un coma (,)

Más sobre los parámetros

```
def saludar(nombre, mensaje="¡Hola!"):
    print(mensaje, nombre)
```

Funciones propias de Python

- Son funciones que tiene Python de forma nativa.
- Cada una tiene su función específica.
- Pueden llamarlas en cualquier parte del código.
- Incluso pueden usar estas funciones dentro de las funciones personalizadas.

Funciones propias de Python

Nombre	Funcionamiento	Ejemplo
abs()	Devuelve el valor absoluto de cualquier número	abs(x)
bin()	Convierte un número entero a binario. Esta función añade el prefijo 'ob' al número binario	bin(x)
enumerate()	Itera sobre un arreglo de datos y al mismo tiempo registra la posición el índice de cada objeto dentro del arreglo	enumerate(x)
len()	Devuelve la longitud de un arreglo o secuencia de datos	len(x)
max(), min()	Devuelve el valor máximo o mínimo de un arreglo de datos	max(x), min(x)
print()	Despliega la información que esté dentro de los paréntesis	print(x)

Tips para las funciones

- Definan muy bien y antes de comenzar qué hará su función exactamente.
- Procuren documentar bien su código, nunca saben quién leerá su código en el futuro,
- Elijan un nombre acorde al funcionamiento de la función.
- Prueben con muchos ejemplos, traten de que su función se infalible.
- Mantengan su código limpio y ordenado.
- Si su función hará dos o más cosas distintas, mejor divídanlas. Divide y vencerás.

Práctica Funciones

Librerías

Aumentando el poder de Python

- Python puede aumentar sus funciones a través de librerías externas.
- Estas librerías son gratuitas y hay para muchos propósitos.
- Las librerías no vienen instaladas, hay que instalarlas para poder utilizarlas.
- En nuestro caso, Google Colab tiene una gran cantidad de librerías ya instaladas que podemos usar
- Para poder utilizar las librerías debemos llamarlas con la palabra clave **import**
- Si quieren conocer cuáles librerías tienen instaladas puede usar el comando **!pip list**.

Numpy

- Es un librería especializada para el cómputo numérico científico y avanzado.
- Es ideal para trabajar con matrices y arreglos de datos.
- Tiene integrada operaciones de álgebra lineal, matricial, transformadas de Fouries, logaritmos, exponenciales, entre otras.
- También pueden utilizar constantes matemáticas como el valor de Pi o el valor de Euler
- Su sintaxis y varias funciones son muy parecidas a Matlab

Matplotlib

- Es una librería que les permite realizar gráficas elegantes, entendibles y personalizadas.
- Pueden hacer gráficas de barras, histogramas, gráficas de pastel, gráficas de dispersión, entre otras.
- Pueden editar vía comandos los aspectos como el título, los ejes, las leyendas, escala de la gráfica, color de líneas, etc.
- En pocas palabras, programan su gráfica para que se vea como ustedes quieran.

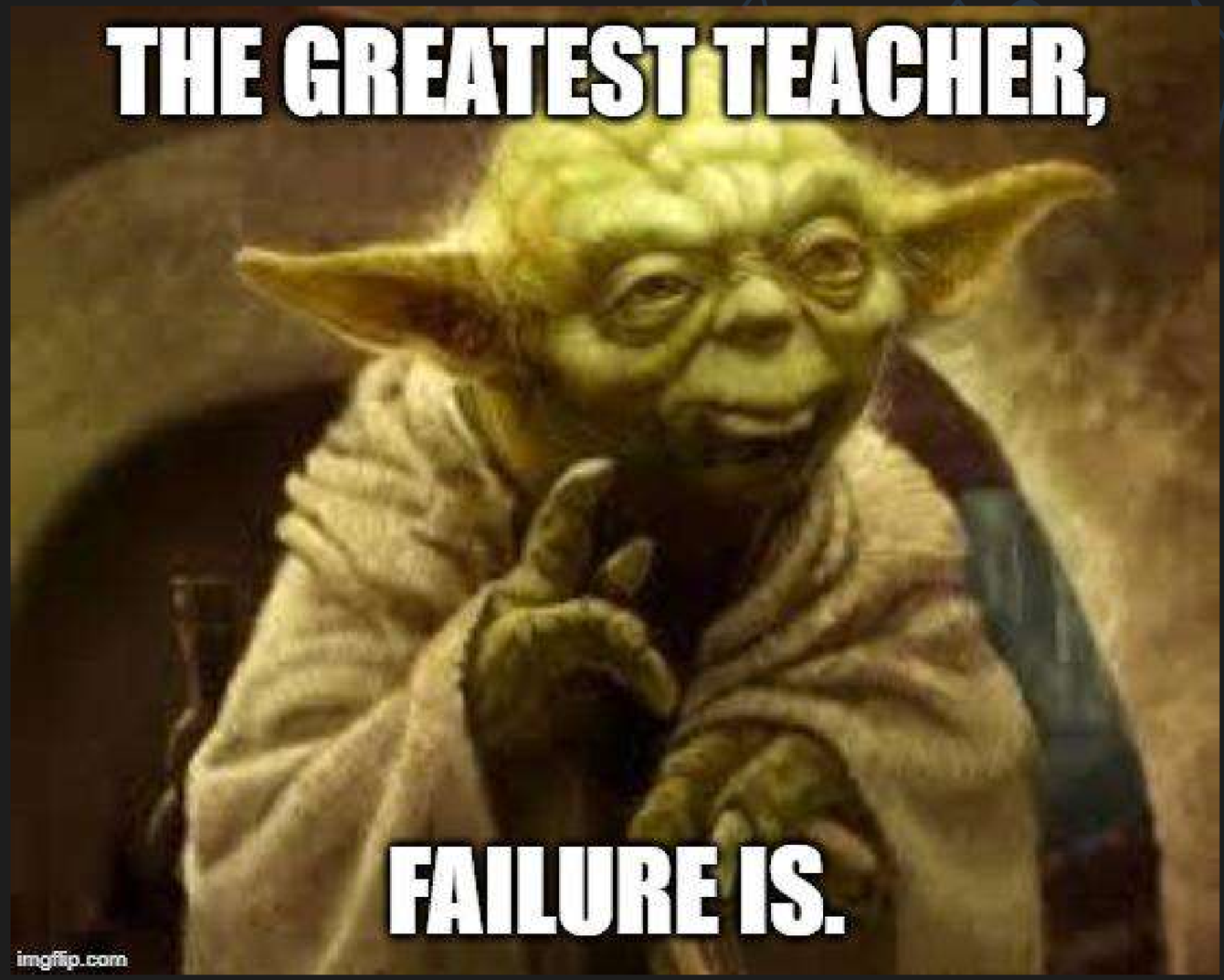
Pandas

- Permite manejar bases de datos y arreglos de datos mas facil y mas ordenado
- Pandas usa dos tipos de datos principales, DataFrame (la base de datos total) y Series (columnas, filas o intervalos).
- Sirve para realizar limpieza de datos, filtrado, ordenamiento, etc.
- Permite leer y escribir bases de datos en formatos de Excel, SQL, csv, JSON, entre otras.
- Puede ejecutar cálculos estadísticos como media, moda, mediana, desviación estándar, cuartiles, etc.

Seaborn

- Librería enfocada a la visualización de datos.
- Permite hacer gráficas mas avanzadas para facilitar interpretabilidad.
- Puede trabajar con Pandas fácilmente.
- Permite un nivel más amplio de personalización

Ya para terminar


THE GREATEST TEACHER,
FAILURE IS.
imgflip.com

Bibliografía

- From zero to hero, José Portilla, Udemy, 2019
- Learning to program: the fundamentals, University of Toronto, Coursera, 2016.
- Python tutorials, w3schools.com
- Python Docs, python.org

Datos de Contacto

- Correo: amhrdz.1001@gmail.com
- Telegram: @amhrdz1001