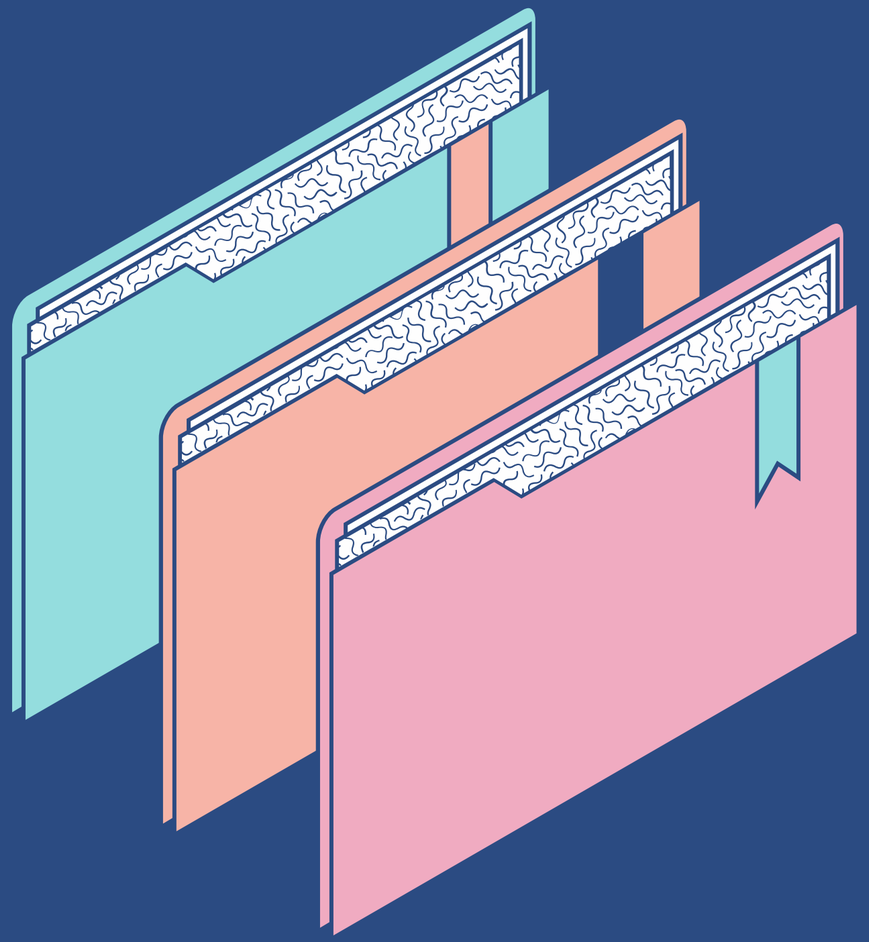
CLASE 19 DE JUNIO

Estructuras en python. Listas, tuplas, diccionarios y conjuntos

Python desde Cero - NineHub



Contenido

- Estructuras de Datos
- Listas
- Tuplas
- Diccionarios
- Conjuntos

Tip: Use links to go to a different page inside your presentation.
How: Highlight text, click on the link symbol on the toolbar, and select the page in your presentation you want to connect.

¿Qué son las estructuras de datos en python?

En Python, las estructuras de datos son objetos que permiten almacenar y organizar datos de manera eficiente para realizar diversas operaciones. Algunas estructuras son las listas, tuplas, diccionarios y conjuntos.

Cada estructura de datos tiene características y métodos específicos que la hacen adecuada para diferentes tipos de operaciones y situaciones.

Listas

[, ,]

Listas

Es una estructura de datos que almacena una **lista** de valores.

Puede almacenar elementos de diferentes tipos (int, float, string, etc.)

Sintaxis: **[, , , ,]**

```
precios = [87, 64, 33, 95, 76]
elementos = [”Alemania”, 85,“asdlfasdfljas”]

elementos[0] #muestra el elemento en posición 0

elementos[0] = “Francia” #cambia el elemento en posición 0

len(elementos) #devuelve la longitud
```

Slicing de listas

Puedes obtener segmentos de una lista preexistente de la siguiente forma:

```
precios = [87, 64, 33, 95, 76]

precios[ 1 : 4 ] → [64, 33, 95]

precios[ : 4 ] = precios[ 0 : 4] → [87, 64, 33, 95]

precios[ 1 : ] = precios[ 1 : len(marks) ] → [64, 33, 95,76]

precios[ -3 : -1 ] → [33, 95]
```

Métodos de listas

```
list = [2, 1, 3]

list.append(4) #añade un elemento al final   [2, 1, 3, 4]

list.sort() #ordena ascendentemente   [1, 2, 3]

list.sort(reverse = True) #ordena descendentemente  [3, 2, 1]

list.reverse() #revierte el orden [3, 1, 2]

list.insert(index, el) #inserta un elemento el en el indice index

list.insert(1, 5)    [3, 5, 1, 2]
```

Métodos de listas

```
list = [2, 1, 3, 1]

list.remove(1) #elimina la primera aparición del elemento
                                                    [2, 3, 1]

list.pop(index) #elimina elemento en el índice index
                          list.pop(2) →  [2, 1, 1]
```

Hay otros métodos. Pueden verse en la documentación.

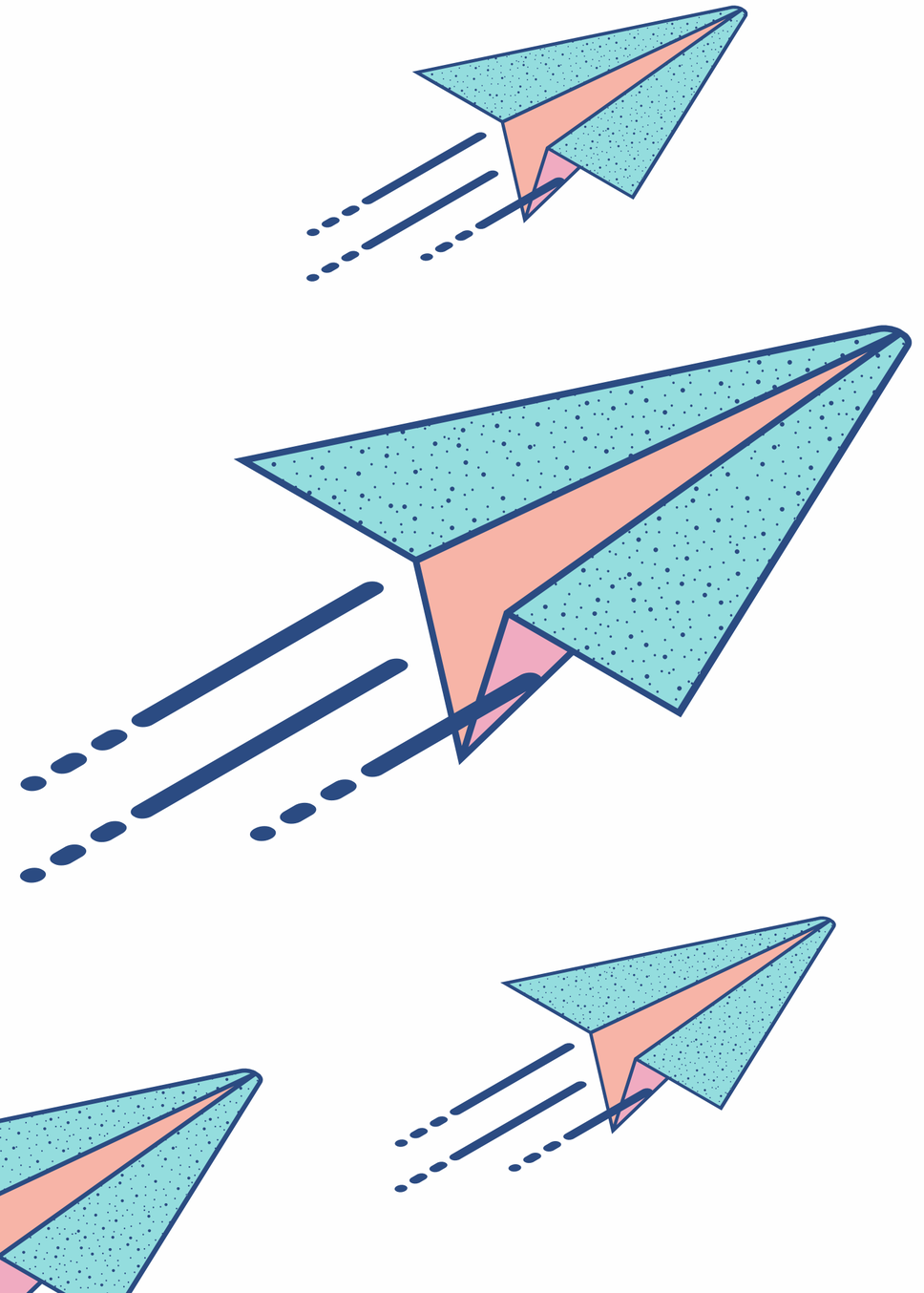
Ejercicio en Clase

- Pedir al usuario que ingrese sus 3 películas favoritas y almacenarlas en una lista. Luego imprimir la lista.

NOTA: Para interactuar con el usuario puedes usar el comando:

```
input(“Escribe tu película favorita”)
```

y luego escribirla.



Solución

```python
lista = [] #Creacion de una lista vacia

while len(lista) < 3:
    pelicula_favorita = input("Ingrese una de sus peliculas favoritas\n")
    lista.append(pelicula_favorita) #Se almacena en la lista las peliculas que haya ingresado

print("Sus Peliculas favoritas son:") #Recorremos la lista con un ciclo for, imprimimos cada elemento en la lista
for pelicula in lista:
    print(pelicula)
```

```
Ingrese una de sus peliculas favoritas
Titanic
Ingrese una de sus peliculas favoritas
Top Gun
Ingrese una de sus peliculas favoritas
Transformers
Sus Peliculas favoritas son:
Titanic
Top Gun
Transformers
```

Tuplas
(, ,)

Tuplas

Es una estructura de datos que almacena una **secuencia inmutable de valores**

Sintaxis: `(,,,,)`

```
tupla = (87, 64, 33, 95, 76)

tupla[0] #muestra el elemento en posición 0

tupla[0] = 245 #NO es permitido en Python
```

Métodos de tuplas

```
tup = (2, 6, 3, 6)

tup.index( el ) #devuelve el índice de la primera ocurrencia
            tup.index(6) → 1

tup.count( el ) #cuenta la cantidad de ocurrencias
            tup.count(6) → 2
```

Hay otros métodos. Pueden verse en la documentación.

Ejercicio en Clase

- Crear una **lista de tuplas** que contenga información sobre varios estudiantes, donde cada tupla contenga el **nombre** del estudiante y su **edad**. Realizar las siguientes operaciones:
  a. Crear la lista de tuplas con al menos tres estudiantes.
  b. Imprimir la lista de estudiantes.
  c. Iterar sobre la lista e imprimir el nombre y la edad de cada estudiante en una nueva línea.

Solución

```python
#a)
estudiantes = [("Ana", 20), ("Juan", 22), ("María", 21)]

#b)
print("Lista de estudiantes:", estudiantes)

# c)
for estudiante in estudiantes:
    nombre = estudiante[0]
    edad = estudiante[1]
    print(f"Nombre: {nombre}, Edad: {edad}")
```

```
Lista de estudiantes: [('Ana', 20), ('Juan', 22), ('María', 21)]
Nombre: Ana, Edad: 20
Nombre: Juan, Edad: 22
Nombre: María, Edad: 21
```

Conjuntos

{ , , }

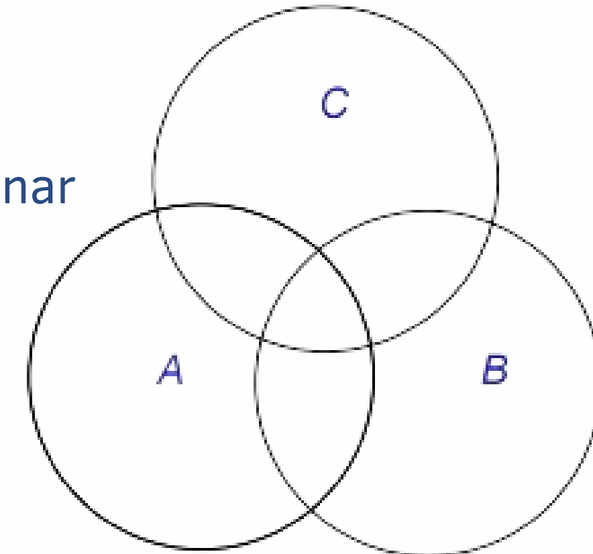
Conjuntos (sets)

Los conjuntos son una estructura de datos fundamental que permite almacenar colecciones de elementos **únicos y no ordenados.** Algunas características importantes de los conjuntos son las siguientes:

- Los conjuntos **no permiten elementos duplicados**. Si intentas añadir un elemento que ya está presente en el conjunto, este no se añadirá de nuevo.

- A diferencia de las listas y las tuplas que mantienen el orden de los elementos según su inserción, **los conjuntos no garantizan ningún orden** específico de los elementos.

- Los conjuntos **son mutables,** lo que significa que puedes añadir y eliminar elementos después de su creación.



¿Cómo creo conjuntos en python?

SE PUEDE DECLARAR CONJUNTOS DE DISTINTAS FORMAS. LA MÁS USUAL ES UTILIZAR CORCHETES “{}”. LOS ELEMENTOS DENTRO DE LOS CORCHETES SERÁN LOS ELEMENTOS DEL CONJUNTO. O BIEN PUEDES CONVERTIR OTRA ESTRUCTURA (COMO UNA LISTA) A UN CONJUNTO.

```
Miconjunto = {1,2,3}
```

```
>>> type(Miconjunto)
<class 'set'>
```

```
lista = [1,2,3]
conjunto = set(lista)
```

```
>>> type(lista)
<class 'list'>
>>> type(conjunto)
<class 'set'>
```

Métodos de Conjuntos

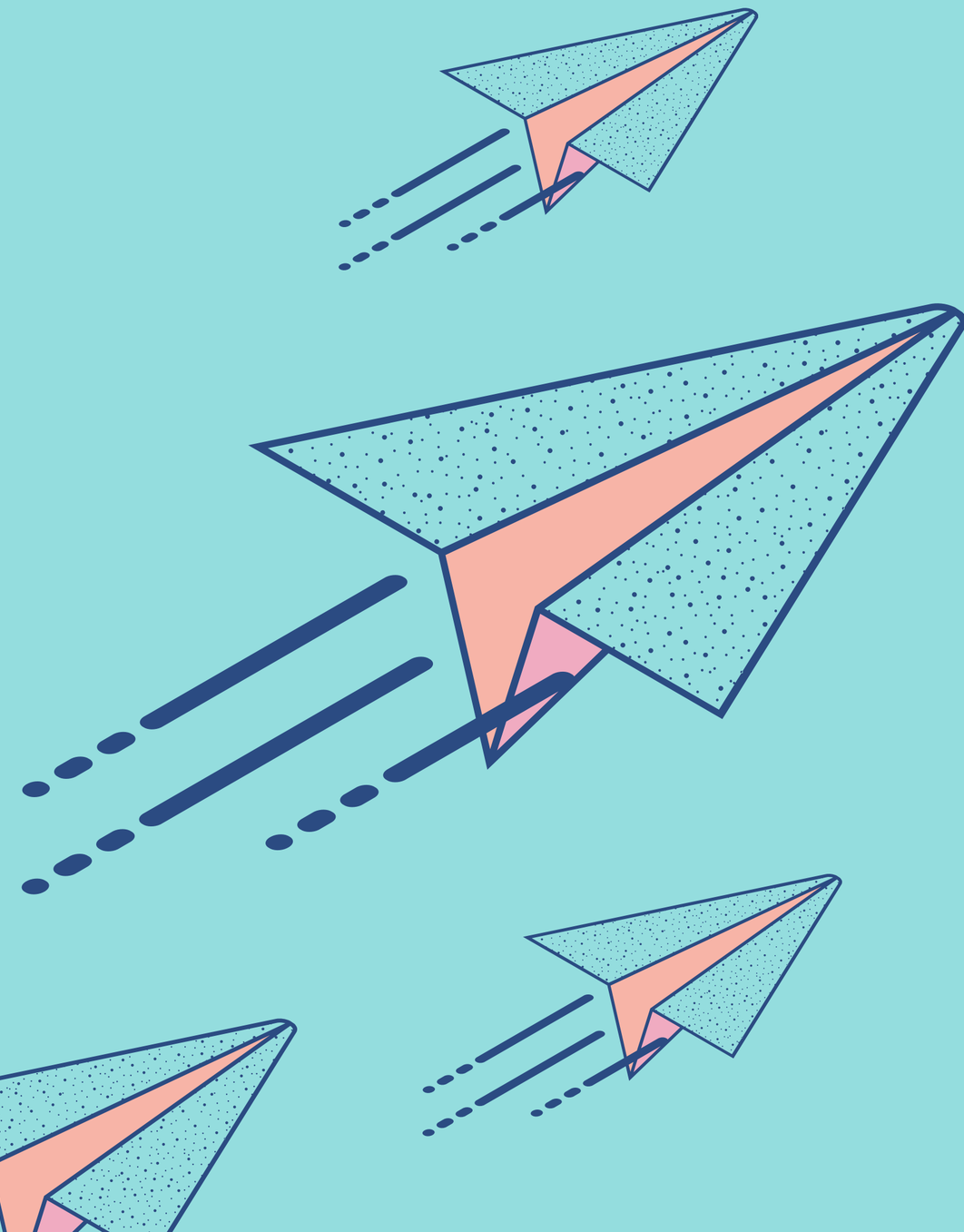
```
set = {1,2,3}
set2 ={1,”a”, “b”}

set.add(element): #agrega un elemento al conjunto {1,2,3,4}
set.remove(element) #elimina un elemento del conjunto  {1, 2}
set.Clear() #Elimina todos los elementos del conjunto, dejándolo vacío {}
set.Pop() #Retorna y luego elimina un elemento aleatorio del conjunto 3
set.union(set2) #Regresa la unión de dos conjuntos {1,2,3,“a“,“b“}
set.insersection(set2) #Regresa la intersección entre dos conjuntos {1}
```

Ejercicio en Clase

Para la siguiente lista, elimina los valores duplicados haciendo uso de los conjuntos. Luego crea un conjunto con números del 1 al 5 y haz la intersección entre ambos conjuntos.

lista = [3,7,5,8,1,6,1,6,2,5]



Solución

```
lista = [3,7,5,8,1,6,1,6,2,5]

lista_sin_duplicados = set(lista)
print("la lista sin duplicados es",list(lista_sin_duplicados))

set2 = {1,2,3,4,5}
interseccion = set2.intersection(lista_sin_duplicados)

print("la interseccion con el conjunto 2 es",interseccion)
```

```
la lista sin duplicados es [1, 2, 3, 5, 6, 7, 8]
la interseccion con el conjunto 2 es {1, 2, 3, 5}
```

Diccionarios

{ : ,
: ,
: }

Diccionarios

Los diccionarios son una estructura de datos fundamental y versátil que permite almacenar pares clave-valor. Son muy utilizados debido a su eficiencia para realizar búsquedas, inserciones y eliminaciones de elementos basados en una clave única. Algunas consideraciones sobre los diccionarios son las siguientes:

- Cada elemento en un diccionario consiste en un par clave-valor, donde la clave es única dentro del diccionario y se utiliza para acceder al valor asociado.

- Los diccionarios son estructuras de datos mutables, lo que significa que puedes modificar, añadir y eliminar elementos después de su creación.

- Similar a los conjuntos, los diccionarios no garantizan ningún orden específico de las claves o valores.

- Las claves de un diccionario deben ser objetos inmutables como cadenas de texto, números o tuplas.

- Los valores en un diccionario pueden ser de cualquier tipo de datos, incluidos otros diccionarios, listas, conjuntos, tuplas, números, cadenas, booleanos, etc

¿Cómo creo diccionarios en python?

SE PUEDEN CREAR DICCIONARIOS USANDO LLAVES CON PARES CLAVE-VALOR O BIEN PUEDES UTILIZAR LA FUNCIÓN DICT CON PARES CLAVE-VALOR COMO TUPLAS, ES DECIR ALGO COMO UNA LISTA DE TUPLAS, SE PUEDE CONVERTIR A UN DICCIONARIO USANDO LA FUNCIÓN DICT.

```
mi_diccionario = {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid'}
```

```
mi_diccionario = dict([('nombre', 'Juan'), ('edad', 30), ('ciudad', 'Madrid')])
```

```
>>> mi_diccionario
{'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid'}
>>> type(mi_diccionario)
<class 'dict'>
```

Métodos de Diccionarios

```
myDict.keys() #Regresa todas las llaves del diccionario

myDict.values() #Regresa todos los valores del diccionario

myDict.items() #Regresa todas las clave-valor como tuplas

myDict.get("key") #Regresa los valores de esa clave del
diccionario

myDict.update(newDict) #Incorpora los elementos especificados
al diccionario.
```

Ejemplos de uso, métodos de diccionarios

```
myDict = {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid'}

print(myDict.keys())
print(myDict.values())
print(myDict.items())
print("El valor en la llave nombre es "+myDict.get("nombre"))

myDict.update({"Sexo": "Masculino"})
print(myDict)
```

```
dict_keys(['nombre', 'edad', 'ciudad'])
dict_values(['Juan', 30, 'Madrid'])
dict_items([('nombre', 'Juan'), ('edad', 30), ('ciudad', 'Madrid')])
El valor en la llave nombre es Juan
{'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid', 'Sexo': 'Masculino'}
```

Ejercicio en Clase

Crea un diccionario de listas que contenga los nombres y calificaciones de los 5 alumnos en la sección de matemática. Luego realiza una búsqueda para conocer el nombre y calificacion de cada alumno en el diccionario.

Solución

```python
# Diccionario con nombres y calificaciones
alumnos = {
    "Nombres": ["Juan", "Pablo", "Ana", "Alejandra", "Julio"],
    "Calificaciones": [92, 84, 75, 89, 100]
}

# Mostrar el diccionario de alumnos y calificaciones
print("Sistema de Alumnos y Calificaciones:")
for i in range(len(alumnos["Nombres"])):
    nombre = alumnos["Nombres"][i]
    calificacion = alumnos["Calificaciones"][i]
    print(f"{nombre}: {calificacion}")
```

```
Sistema de Alumnos y Calificaciones:
Juan: 92
Pablo: 84
Ana: 75
Alejandra: 89
Julio: 100
```

Para finalizar, veamos esta pequeña comparación de las estructuras vistas en clase

Característica	LISTA	TUPLA	CONJUNTO	DICCIONARIO
Mutabilidad	Mutable	**Inmutable**	Mutable	Mutable
Orden	Ordenada	Ordenada	**No ordenado**	**No ordenado**
Índices	Por índice (enteros)	Por índice (enteros)	No tiene índices	**Por clave**
Sintaxis	**[ ]**	**( )**	**{ }**	**{ key: value, ... }**
Ejemplo	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{'a': 1, 'b': 2}
Uso común	Almacenar colecciones de elementos modificables	Datos que no deben modificarse	Eliminar duplicados o pruebas de membresía rápidas	Mapear claves a valores