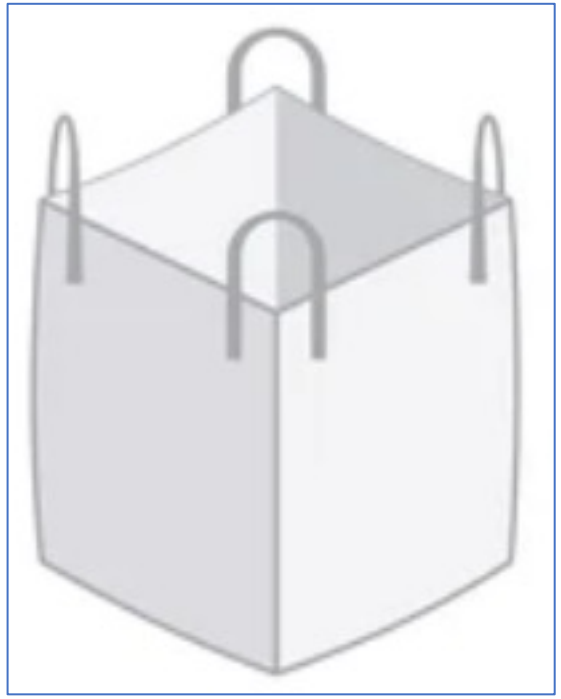
Utilización Contenedores

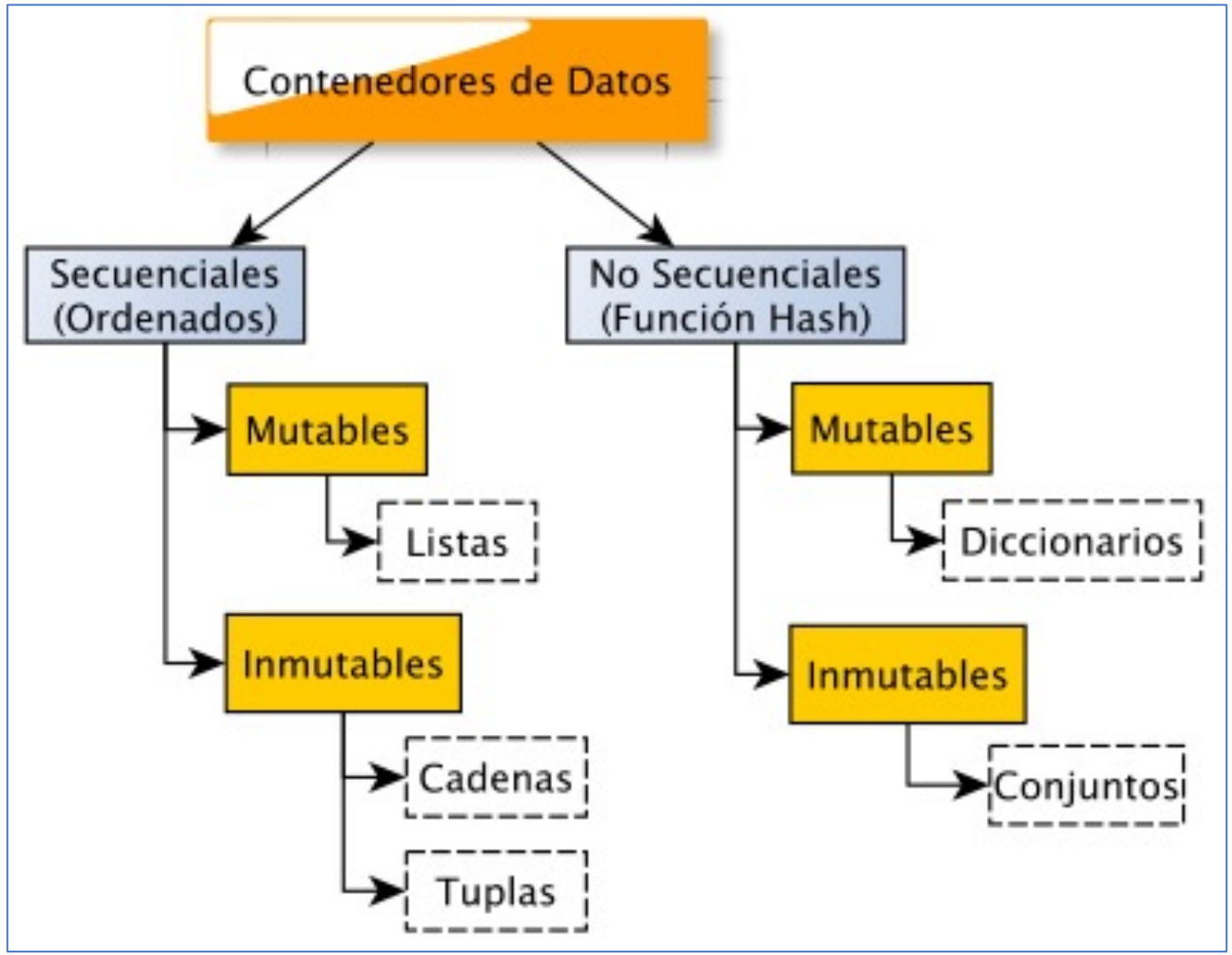
Listas, Conjuntos,
Tuplas y Diccionarios

Contenedores en Python

- Colecciones, arreglos o contenedores: "Bolsas" para agrupar de diferentes maneras varios datos.
- Técnicamente: Estructuras de Datos.
- Clasificación habitual: mutables e inmutables.
- En este caso priorizaremos la clasificación según su funcionalidad.



Contenedores en Python



Ordenados

- Cuando los elementos deben seguir un orden o una secuencia al ser coleccionados, se recomienda el uso de las **listas, las cadenas y las tuplas.**
- El contenido de los elementos se puede repetir, pero son etiquetados con un índice autonumérico.
- Si la colección no se va a modificar durante el tiempo de ejecución, **las cadenas** para los caracteres y las **tuplas** como equivalentes de listas son las indicadas.
- Si la colección se va a modificar durante el tiempo de ejecución, **las listas** serán la mejor opción.

Ordenados: Strings (Cadenas)

- Utilizamos este contenedor para almacenar flujos de caracteres.
- Una vez almacenado, no se modificará, podrían generarse nuevas cadenas a partir de estos, pero sin actualizaciones.
- **Ejemplo:** Almacenar un atributo alfanumérico para luego hacerlo parte de un flujo que se empleará en un mensaje.

```
cadena = "Este es un atributo "
nuevaCadena = cadena + "nuevo"
print("Flujo: ",nuevaCadena) -> "Flujo: Este es un atributo nuevo"
print(nuevaCadena[0:4]) -> "Este"
```

Ordenados: Tuplas

- Utilizamos este contenedor para almacenar elementos de diferente tipo, que no van a modificarse durante la ejecución del programa.

Ejemplos:

- Almacenamiento de parámetros cargados desde una base de datos o desde un archivo, que servirán como base de los cómputos que se realicen durante un proceso; casos como límites superiores, valores de configuración.

- Envío de información entre módulos que no debe modificarse.

Ordenados: Tuplas

- **Ejemplo 1:** Coleccionar credenciales en una tupla para conexión con BD:

```
def calcularCostoArticulo():
    nombreUsuario = "tripulanteMisionTIC2022"
    password = "pr0gr4m4c10n"
    tuplaCredencialesBD = (nombreUsuario,password)#Tupla
    impuestos = obtenerImpuestosBD( tuplaCredencialesBD )
    .
    .
    .
```

- **Ejemplo 2:** Distancias calculadas entre puntos geográficos:

```
distancias = ( ("Pereira","Bogotá",230) ("Cali","Pereira",260) )
```

Ordenados: Listas

- Utilizamos este contenedor para almacenar elementos de diferente tipo, que van a modificarse durante la ejecución del programa, es decir, podrían agregarse, eliminarse o actualizarse elementos.

Ejemplos:

- Construcción de un itinerario turístico.
- Productos en un carrito de compras.

Ordenados: Listas

- **Ejemplo:** Coleccionar destinos de itinerario turístico y actualizarlo:

```
itinerario = [["Santa Marta",1],["Cartagena",2],["San Andrés",4]]
itinerario.append(["Providencia",2])#Agregar Providencia, 2 noches
itinerario.pop(1)#Eliminar visita a Cartagena, 2 noches
itinerario[0][1] += 1 #Agregar una noche en Santa Marta
#Cambiar el inicio del itinerario por Providencia:
#Dejar Santa Marta para el final
itinerario[0],itinerario[-1] = itinerario[-1],itinerario[0]
print(itinerario) -> [['Providencia', 3], ['San Andrés', 4], ['Santa Marta', 1]]
```

Ordenados: Recorridos

- Todos los contenedores ordenados cuentan con un autonumérico, por lo tanto, pueden recorrerse con un subíndice:

```
#Mostrar posición y contenido de cada elemento
for i in range(len(contenedorOrdenado)):
    print("Elemento en la posición",i)
    print(contenedorOrdenado[i])
```

- O se puede utilizar la carga de cada elemento del iterable en una variable auxiliar:

```
for elemento in contenedorOrdenado:
    print(elemento)
```

Ordenados: Recorridos

- La combinación de los recorridos anteriores, para evitar notación de subíndice, pero tener disponible la posición en cada iteración tenemos **enumerate**:

```
for i,destino in enumerate(itinerario):
    print("Posición: ",i)
    print(destino)
```

```
Posición:  0
['Providencia', 2]
Posición:  1
['San Andrés', 4]
Posición:  2
['Santa Marta', 2]
```

No Secuenciales

- Cuando los elementos no deben seguir un orden o secuencia al ser coleccionados, se recomienda el uso de **conjuntos y diccionarios**.
- Las etiquetas de los elementos que agrupan no se repiten, en tal caso se generaría una actualización del mismo elemento.
- Las etiquetas se deben generar, no son automáticas, pero pueden ser de cualquier tipo de dato
- Si la colección de este tipo no se va a modificar durante el tiempo de ejecución, y es una estructura sin anidaciones, **los conjuntos** serían los indicados.
- Si la colección se va a modificar durante el tiempo de ejecución, y la información presenta una estructura jerárquica, **los diccionarios** serán la mejor opción.

No Secuenciales: Conjuntos

- **Ejemplo:** Conjunto de control de los grupos de MisiónTIC2022 que han sido calificados. Notar que aunque los elementos son inmutables, la estructura tipo conjunto puede crecer o disminuir en número de elementos. Es muy útil para listados de control. Naturalmente prohíbe repeticiones.

```
gruposCalificados = {'P45','P61','P33','P87'}
#Observar el tipo que ha sido identificado
print("Grupos calificados es de tipo: ",type(gruposCalificados))
gruposCalificados.add('P17')
gruposCalificados.add('P17')
gruposCalificados.remove('P45')
print(gruposCalificados) -> {'P61', 'P87', 'P33', 'P17'}
if 'P17' in gruposCalificados: #Revisar pertenencia
    print('P17 ya ha sido calificado!')
```

No Secuenciales: Diccionarios

Uso:

- Almacenar información intrincada para representar entidades del mundo real en nuestro sistema de información con etiquetas (índices o llaves) personalizados.
- En las versiones recientes de Python, se respeta el orden en el que fueron agregados los elementos, sin que esta sea su prioridad.

Ejemplo:

- Cargar información de varios estudiantes, identificados con un código establecido por registro y control de la institución a la cual pertenecen.

No Secuenciales: Diccionarios

```
diccionarioEstudiantes = {
    'E3454fdf':{
        'nombres': 'Laura',
        'apellidos': 'Jaramillo',
        'acudientes': ['Andrea','Juan'],
        'promedio':5.0
    },
    'Egg56dfg':{
        'nombres': 'Samir',
        'apellidos': 'Gómez',
        'acudientes': ['Alejandro','Sofía'],
        'promedio':5.0
    },
    'FHT43523':{
        'nombres': 'Sara',
        'apellidos': 'Cabrera',
        'acudientes': ['Carlos','Amparo'],
        'promedio':5.0
    },
    'Z4edkdf':{
        'nombres': 'Iván',
        'apellidos': 'Arcila',
        'acudientes': ['Esposa'],
        'promedio':5.0,
        123:'Este estudiante es alérgico al maní'
    }
}
```

No Secuenciales: Recorridos

- Los contenedores no secuenciales, no cuentan con autonuméricos, solamente etiquetas, porque no es su objetivo mantener un orden, sólo eficiencia en respuesta (adición, eliminación y consulta).

- Comúnmente se utiliza la carga de cada elemento del iterable, en una variable auxiliar:

```
for elemento in contenedorNoSecuencial:
    print(elemento)
```

No Secuenciales: Recorridos

- En el caso de los no secuenciales, **enumerate** puede servir para mostrar un conteo de los elementos, pero no hay una asociación entre este número y el elemento.

```
for i,grupo in enumerate(gruposCalificados):
    print("Número: ",i)
    print(grupo)
```

```
Número:  0
P61
Número:  1
P33
Número:  2
P87
Número:  3
P17
```

No Secuenciales: Recorridos

- Particularmente, los diccionarios se recorren como **ítems** que presentan una llave y un valor, análogo a lo que se realiza con **enumerate**.

```
for codigoEstudiante,infoEstudiante in diccionarioEstudiantes.items():
    print("Llave",codigoEstudiante)
    print("Valor",infoEstudiante)
```

```
Llave E3454fdf
Valor {'nombres': 'Laura', 'apellidos': 'Jaramillo', 'acudientes': ['Andrea', 'Juan'], 'promedio': 5.0}
Llave Egg56dfg
Valor {'nombres': 'Samir', 'apellidos': 'Gómez', 'acudientes': ['Alejandro', 'Sofía'], 'promedio': 5.0}
Llave FHT43523
Valor {'nombres': 'Sara', 'apellidos': 'Cabrera', 'acudientes': ['Carlos', 'Amparo'], 'promedio': 5.0}
```