



unab

UNIVERSIDAD NACIONAL
GUILLERMO BROWN

Tipos de Datos Definidos por el Usuario

Algóritmos y Estructuras de Datos

-00184-

Dr. Diego Agustín Ambrossio

Anl. Sis. Angel Leonardo Bianco

Overview:

Tipos de Datos:

- Definidos por el lenguaje (simples)
 - Númericos: Enteros, Reales, Complejos,
 - Cadena de Caracteres,
 - LÓGICOS (booleanos).
- Definidos por el Usuario (complejos)
 - Tuplas,
 - Conjuntos,
 - Diccionarios,
 - Listas,
 - Rangos.
- Datos Mutables e Inmutables.
- Punteros.

Tipos de Datos:

- Tipos de Datos Simples.
 - Definidos por el lenguaje.
 - Números:
 - Enteros (**type** ‘int’)
 - Reales (**type** ‘float’)
 - Complejos (**type** ‘complex’)
 - Cadena de Caracteres (**type** ‘str’)
 - Booleanos (**type** ‘bool’)
 - Operadores Relacionales
- Tipos de Datos Complejos
 - Definidos por el Usuario
 - Tuplas (**type** ‘tuple’)
 - Conjuntos (**type** ‘set’)
 - Diccionarios (**type** ‘dicc’)
 - Listas (**type** ‘list’)
 - Rangos (**type** ‘range’)

Tipos de Datos Definidos por el Usuario:

- Un tipo de dato define el posible **rango de valores** que una variable puede tomar al momento de ejecución del programa y a lo largo de toda la vida útil del programa. Este rango esta definido por el usuario.

- Tipos de Datos Definidos por el Usuario:
 - Tuplas (**type** ‘tuple’)
 - Conjuntos (**type** ‘set’)
 - Diccionarios (**type** ‘dicc’)
 - Listas (**type** ‘list’)
 - Rangos (**type** ‘range’)

Además del rango de valores, también podemos definir las operaciones aceptadas
(estructura del tipo de dato)

Tuplas:

Una **tupla** es una secuencia finita de objetos (los cuales pueden ser de distintos tipos).

- Esta es una estructura de datos **rígida**, es decir, no podemos modificar una tupla. (INMUTABLE!)
- Las tuplas pueden ser concatenadas, de manera similar a la concatenación de strings (usando el '+').
- Podemos obtener la logitud de una tupla, utilizando la función `len()`.
- También podemos acceder a los elementos de la tupla usando indices.

Tuplas:

Definicion:

```
1 # Definición de una tupla.  
2 x=(0,1,2,'obj',(1,2))  
3 # Tipo  
4 type(x)
```

No hay muchos métodos específicos asociados a las tuplas.

- **x.count(obj)** retorna el número de occurencias del objeto 'obj' en 'x'.
- **x.index(obj)** retorna el indice de la primer occurencia del objeto 'obj' en 'x'.

Conjuntos

Un **conjunto** es una colección finita de objetos. Los conjuntos tienen (deben cumplir) todas las propiedades de los conjuntos matemáticos.

Definición:

```
1 # Definición de un conjunto.  
2 A={1,2,4,5,6}  
3 # Tipo.  
4 type(A)
```

Conjuntos

Aquí hemos provisto una lista (no completa) de métodos útiles para el manejo de conjuntos:

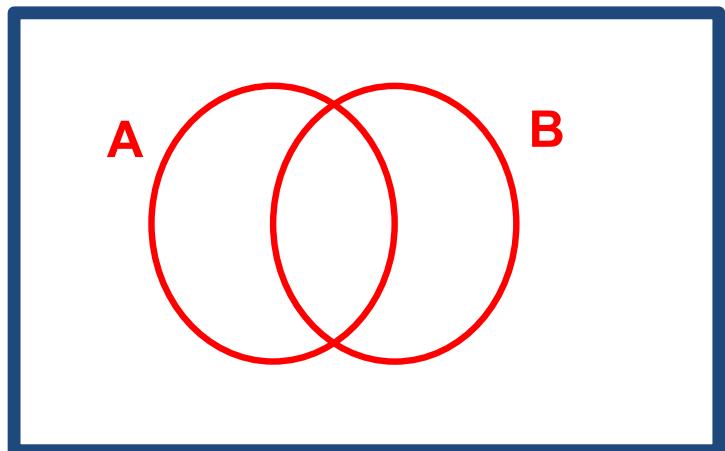
- **A.intersection(B)**: retorna la intersección de los conjuntos A y B.
- **A.union(B)**: retorna la unión de los conjuntos A y B.
- **A.difference(B)**: retorna la diferencia simétrica de conjuntos
- **A.copy()**: retorna una copia del conjunto A.

Podemos comparar dos conjuntos (seguiendo el orden de inclusión).

- **Diagrama de Venn**

Representación gráfica de conjuntos: **Rectángulo → Universo**

Subconjuntos de ese universo → A, B



Universo

Sean los conjuntos:

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}$$

$$A = \{1, 2, 3, 5, 7, 11, 13, 17\}$$

$$B = \{1, 2, 4, 6, 8, 9, 10, 12, 13, 14, 15, 17\}$$

Nota:

Los conjuntos no aceptan repeticiones, es decir:

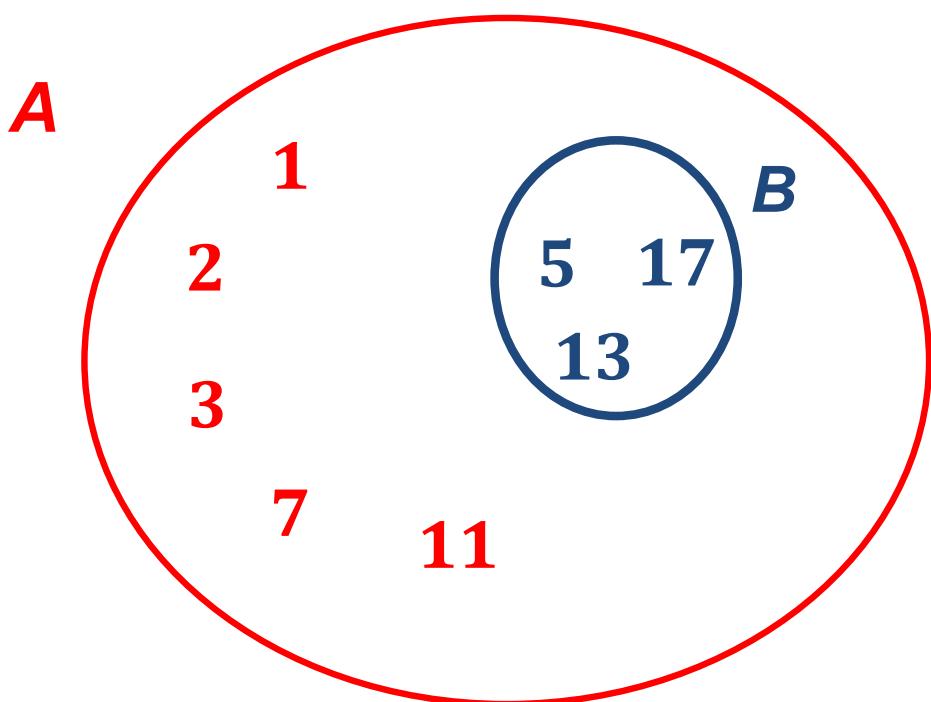
$$A = \{1, 2, 3\}$$

$B_1 = \{1, 2, 2, 3, 1, 3, 4\}$ no es un conjunto, pero $B_2 = \{1, 2, 3, 4\}$ lo es.

- Diagrama de Venn

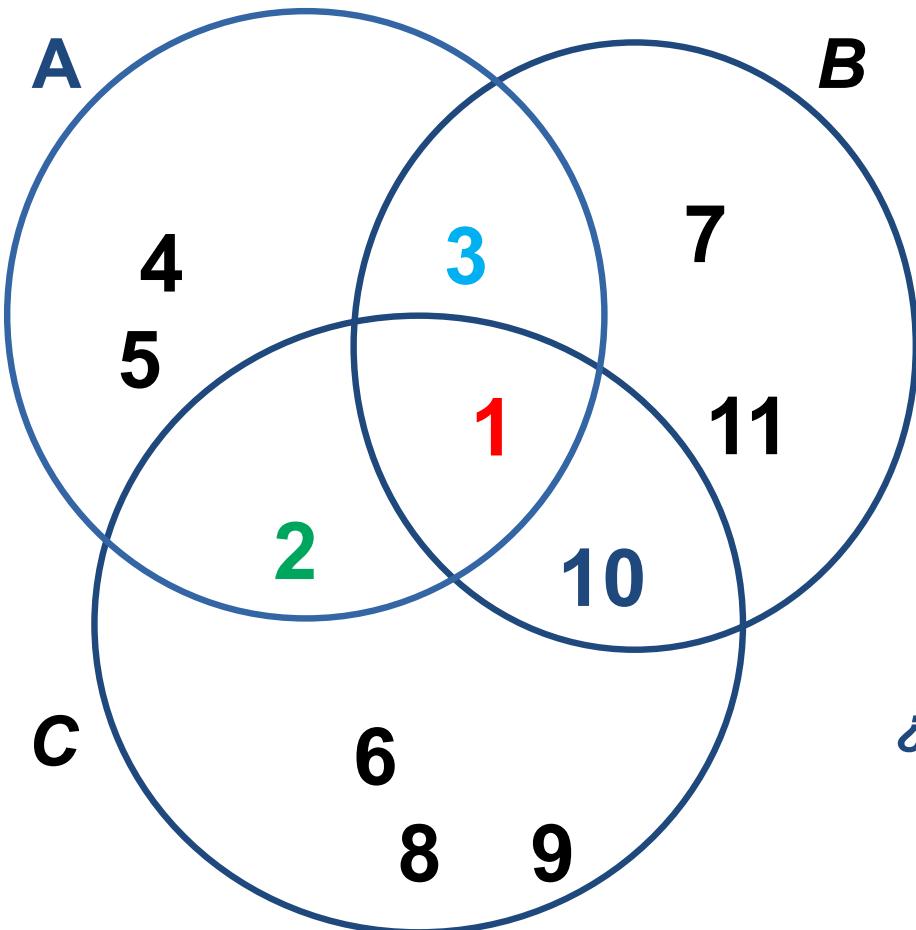
Inclusión de conjuntos:

$A = \{1, 2, 3, 5, 7, 11, 13, 17\}$ y $B = \{5, 13, 17\}$,



se dice que
 $B \subset A$
(está incluido)

- Diagrama de Venn



Sean los conjuntos:

$$A = \{1, 2, 3, 4, 5\}$$

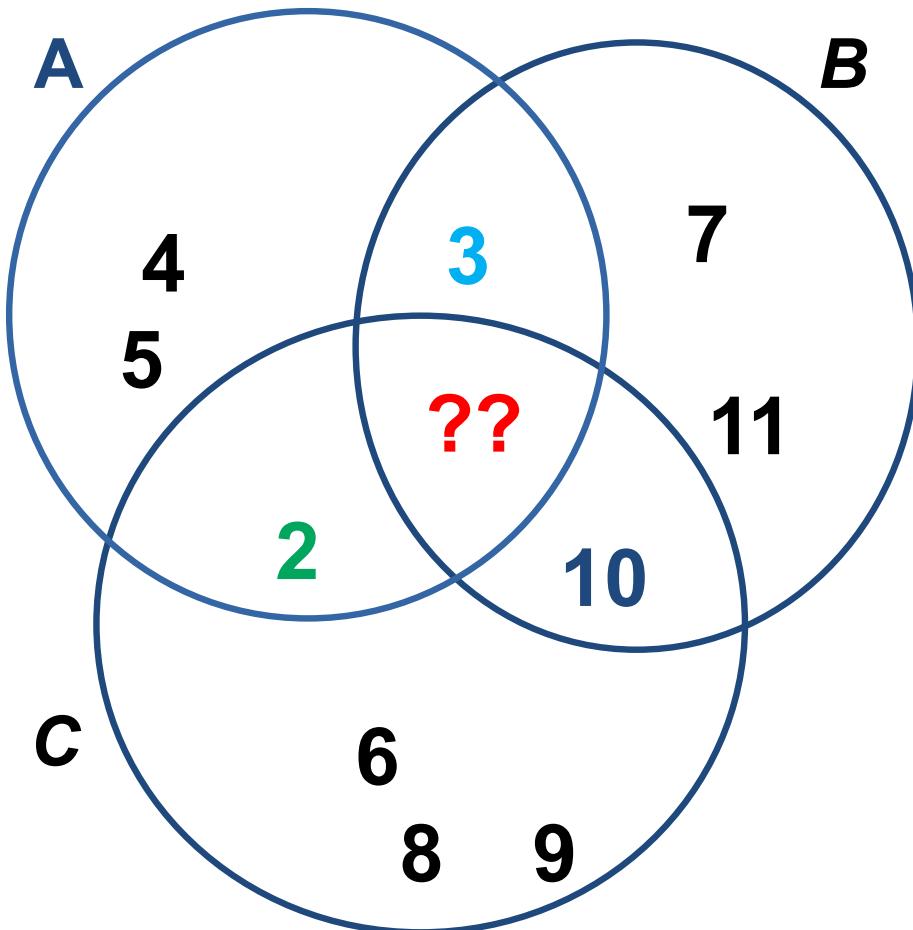
$$B = \{1, 3, 7, 10, 11\}$$

$$C = \{1, 2, 6, 8, 9, 10\}$$

$$A \cap B \cap C = \{1\}$$

¿Cómo es la unión entre A, B y C?

- Diagrama de Venn



Sean los conjuntos:

$$A = \{2,3,4,5\}$$

$$B = \{3,7,10,11\}$$

$$C = \{2,6,8,9,10\}$$

¿Cómo es la unión entre A, B y C?

$$A \cap B \cap C = \{\}$$

Alternativamente podemos escribir \emptyset

$$A \cap B \cap C = \emptyset$$

Conjuntos

También podemos realizar cambios directamente sobre un conjunto:

- **A.add(a)**: reemplaza el conjunto A por A union a (no tiene efecto si el elemento a pertenece al conjunto A).
- **A.discard(a)**: reemplaza el conjunto A por A removiendo el elemento a (no tiene efecto si el elemento a no pertenece al conjunto A).
- **A.difference_update(B)**: reemplazara el conjunto A por A intersección B^C.

Diccionarios:

- Un **diccionario** o "tabla asociativa" es una colección de items "llave : valor", donde la **llave** y el **valor** pueden ser cualquier tipo de objeto.

- y los escribimos de la siguiente forma:

c = {l1:v1, l2:v2, ..., ln,vn}

```
1 dicc1={'Jean Paul':'jeanpaul@mail.co
2 'Fanny':'fanny@mail.com',\
3 'Robert':'robert@mail.com',\
4 'Stephanie': (6812424239),\
5 0:2}
6
7 # Tipo
8 print(type(dicc1))
```

Es un contenedor de amplio uso (y estan optimizados al igual que cualquier tipo "hashable" en Python).

Diccionarios:

- La función `len()` nos devuelve la cardinalidad.
- No podemos indexarlos de manera habitual, es decir de manera posicional, deberemos usar las llaves de diccionario para retornar los valores correspondientes.
- Es posible asignar nuevos valores a una las llaves.

`dic[key]=value` asignará un nuevo valor a la llave `key`, solo si la llave ya pertenece al diccionario.

En caso de que la llave no exista en el diccionario, añadira el nuevo par `(key:value)` al diccionario.

Diccionarios:

Aquí tenemos algunos métodos útiles para usar diccionarios.

- **dic.item()**: retorna una *lista* de los valores en el diccionario
- **dic.keys()**: solo retorna los valores de las llaves del diccionario.
- **dic.values()**: solo retorna los valores del diccionario.
- **dic.copy()**: retorna una copia del diccionario \$\texttt{dic}\$.
- **dic.pop(key)**: retorna y quita del diccionario el valor que referido por la llave **key**
- **dic.popitem()** retorna y quita del diccionario el último item ingresado/añadido.
- **dic.update(newdic)** actualiza **dic** con los valores de otro diccionario **newdic**, solo para las llaves de **newdic** que no están en **dic**.

Listas:

Una **lista** es un contenedor ordenado de objetos (pueden ser de distinto tipo).

Definicion:

```
1 # Definición
2 L=[2,3,4]
3 # Tipo
4 type(L)
```

Definición por comprensión:

```
1 # También podemos definir Listas por comprensión (al igual que los conjuntos).
2 L=[x**2 for x in range(0,9)]
3 L
```

Listas:

Podemos utilizar los indices de distintas maneras para poder acceder mas convenientemente a partes de una lista. Esta notación funciona para cualquier tipo de contenedor ordenado (i.e. strings, tuples, etc).

- **L [i]** retorna el i-esimo elemento de la lista **L**.
- **L [i : j]** retorna los elementos desde el i-esimo (incluido) hasta el j-esimo (excluido). El resultado tiene el mismo tipo que **L**.
- **L [i :]** es equivalente a: **L [i : len (L)]**.
- **L [:j]** es equivalente a: **L [0 :j]**.
- **L [i :: paso]** retorna la lista de elemnts desde la **\$i\$**-esima posición progresando sobre los indices de acuerdo a **paso**.
- También podemos concatenar listas usando el operador **'+'**

Listas:

Aquí hay una breve lista de los métodos asociados al tipo **Lista**:

- **L.count(obj)** retorna el número de ocurrencias del objeto ‘obj’ en L.
- **L.index(value)** retorna el primer indice **i** para el cual **L [i] = value**.
- **L.insert(i, obj)** inserta el objeto **obj** en la i-esima posición, moveindo el resto de los contenidos de la lista hacia la derecha.
- **L.remove(value)** remueve de L la primer ocurrencia de \$\\texttt{value}.
- **L.pop(index)** retorna el valor indexado y lo remueve de L.
- **L.reverse()** escribe la lista L del revez (cambia L).
- **L.sort(L)** (re)ordena L de acuerdo al orden lexicográfico. Los elementos deben ser del mismo tipo.

Rangos:

- Los **rangos** son tipos especiales de contenedores.

Construiremos un **rango** de la siguiente forma:

`range(comienzo, fin, paso)`

- El rango no es un objeto **per se**, podemos tratarlo como una '*lista potencial*'.
- Es posible checkear si 'valor' esta (o pertenece) a cierto rango, i.e. 'valor' `in range(comienzo, fin)`

Diferencias entre Contenedores:

- Un **diccionario** es una forma conveniente de almacenar/actualizar/eliminar información sobre llaves específicas. Sin embargo, es un objeto de mayor complejidad y debe ser utilizado con cuidado.

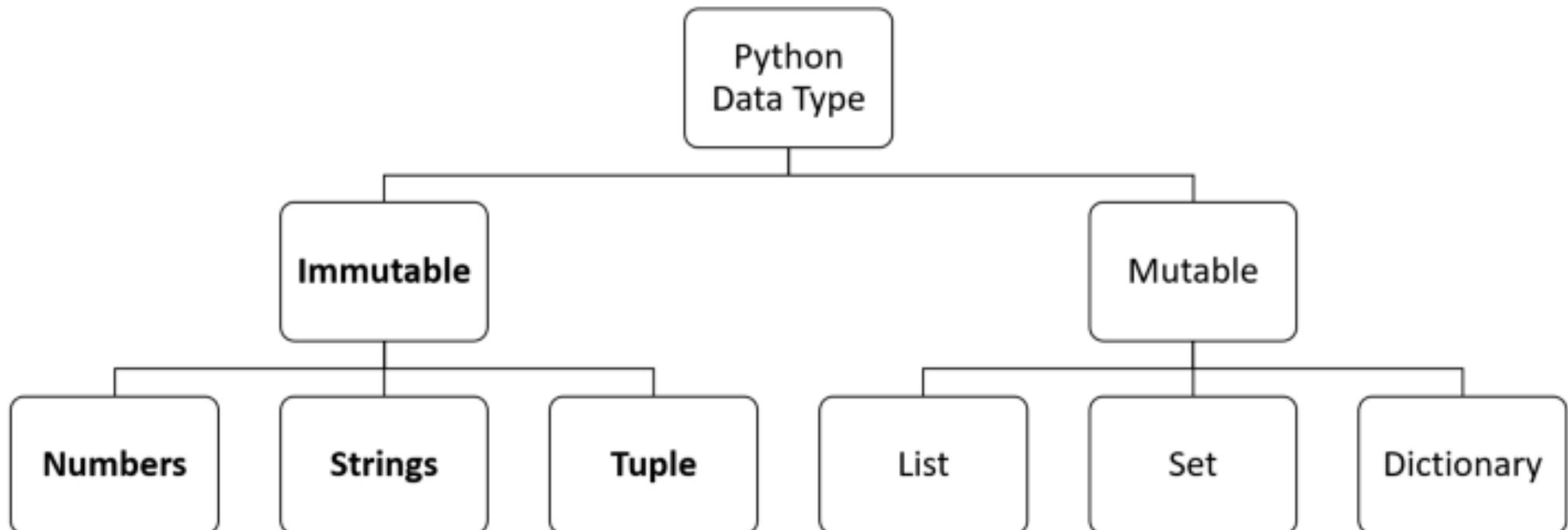
- Una cadena de caracteres es un objeto específico. Es una forma simple se comunicarse con el usuario, usando el método format.

- Los **strings** y las **tuplas**, son tipos inmutables: es decir,
No existen métodos definidos para cambiar sus valores internos.

Diferencias entre Contenedores:

- Los **conjuntos, diccionarios y listas** son tipos "mutables".
 - Existen diversos métodos para manipularlos.
 - Debemos tener cuidado cuando utilizamos la asignación en tipos mutables, ya en generalmente sobrescribiremos el o los valores.
 - Para copiar un objeto complejo debemos usar el método `copy` en vez del simbolo `=`.
- Los objetos "mutables" tiene un tiempo de acceso mayor (son un poco más lentos).
- La desventaja de usar tipos "mutables" es el acceso más lento a los datos.

Diferencias entre Contenedores:



Punteros:

- En otros lenguajes, existe un tipo de variable especial llamado **puntero**, que se comporta como una referencia a una variable (**como es el caso de las variables mutables en Python**).
- En Python **no hay punteros* de la manera tradicional*, como en la mayoría de los lenguajes OO, todas las variables son referencias a objetos.
Es decir, la **dirección** de la porción de memoria, en donde el objeto está almacenado.
Si esa porción de memoria cambia, el cambio se puede ver en todas las variables que apuntan a esa porción.
- Ejemplo `type('obj')`

Vectores y Matrices:

Algoritmos y Estructuras de Datos

unq



Algoritmos y Estructuras de Datos



Algoritmos y Estructuras de Datos



Algoritmos y Estructuras de Datos



Algoritmos y Estructuras de Datos

