TEMA II

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"Un sueño no se hace realidad por arte de magia, necesita sudor, determinación y trabajo duro"
Colin Powell

3. Tipos de datos secuencia (contenedores)

Me permiten guardar una sucesión de datos de diferentes tipos. Los tipos de datos secuencias en python son:

- Las listas (list): guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos. Es un tipo mutable.
- Las tuplas (tuple): Sirven para los mismo que las listas, pero en este caso es un tipo inmutable.
- Los rangos (range): nos permite crear secuencias de números. Es un tipo inmutable y se suelen utilizar para realizar bucles.
- Los conjuntos (set): guardar conjuntos de datos en los que no existen repeticiones. Es un tipo mutable.
- Los conjuntos (frozenset): Son iguales que los anteriores pero inmutables.

3.1 Listas

Colecciones que representan secuencias ordenadas de objetos de distintos tipos. Se representan con corchetes y los elementos se separan por comas.

Ejemplo. [1, "dos", [3, 4], True].

Se caracterizan por:

- Tienen orden.
- Pueden contener elementos de distintos tipos.
- Son mutables, es decir, pueden alterarse durante la ejecución de un programa.

No es el único tipo de dato compuesto pero sí el más versátil.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Al igual que las cadenas, se pueden indexar y segmentar:

```
>>>
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

Todas las operaciones de indexación retornan una nueva lista que contiene los elementos pedidos.

Creación de listas mediante la función list()

Otra forma de crear listas es mediante la función list(). Crea una lista con los elementos de la secuencia o colección.

```
>>> list()
[]
>>> list("Python")
['P', 'y', 't', 'h', 'o', 'n']
```

Para convertir una lista de string en una cadena utilizamos la función join

a.join(b) concantena los elementos de la lista b insertando la cadena a entre ellos

Ejemplo

```
nom=list("rosa")
nom[0]="R"
nombre="".join(nom)
print(nombre)
```

Sublistas

l[i:j:k] : Devuelve la sublista desde el elemento del con el índice i hasta el elemento anterior al índice j, tomando elementos cada k.

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> a[1:4]
['y', 't', 'h']
>>> a[1:1]
[]
>>> a[:-3]
['y', 't', 'h']
>>> a[:]
['P', 'y', 't', 'h', 'o', 'n']
>>> a[0:6:2]
['P', 't', 'o']
```

Operaciones que no modifican una lista

- len(l): Devuelve el número de elementos de la lista
- min(l): Devuelve el mínimo elemento de la lista siempre que los datos sean comparables.
- max(l): Devuelve el máximo elemento de la lista siempre que los datos sean comparables.
- sum(l): Devuelve la suma de los elementos de la lista, siempre que los datos se puedan sumar.
- dato in l: Devuelve True si el dato pertenece a la lista
- l.index(dato): Devuelve la posición que ocupa en la lista el primer elemento con ese valor.

➢l.count(dato): Devuelve el número de veces que el dato está contenido en la lista

➢all(l): Devuelve True si todos los elementos de la lista y False en caso contrario.

➢any(l): Devuelve True si algún elemento de la lista es True y False en caso contrario.

```
>>> a = [1, 2, 2, 3]

>>> 3 in a
True
>>> a.index(2)
1



>>> a.count(2)
2
>>> all(a)
True
>>> any([0, False, 3<2])
False
```

Operaciones que modifican una lista

➢l1 + l2: Crea una nueva lista concatenan los elementos de la listas

➢l.append(dato): Añade dato al final de la lista

➢l.extend(sequencia) : Añade los datos de secuencia al final de la lista

➢l.insert(índice, dato): Inserta dato en la posición índice de la lista y desplaza los elementos una posición a la derecha

➢l.remove(dato): Elimina el primer elemento con valor dato y desplaza los que están por detrás de él una posición hacia delante. Si no existe da error al ejecutar.

➢l.pop([índice]): Devuelve el dato en la posición índice y lo elimina desplazando los elementos por detrás de él una posición hacia delante.

➢l.sort(): Ordena los elementos de la lista de acuerdo al orden predefinido, siempre que los elementos sean comparables.

➢l.reverse() : invierte el orden de los elementos de la lista

```
>>> a = [1, 3]
>>> b = [2 , 4, 6]
>>> a.append(5)
>>> a
[1, 3, 5]
>>> a.remove(3)
>>> a
[1, 5]
>>> a.insert(1, 3)
>>> a
[1, 3, 5]
>>> b.pop()
6
>>> c = a + b
>>> c
[1, 3, 5, 2, 4]
>>> c.sort()
>>> c
[1, 2, 3, 4, 5]
>>> c.reverse()
>>> c
[5, 4, 3, 2, 1]
```

Copia de listas

Existen dos formas de copiar listas:

Copia por referencia l1 = l2: Asocia la la variable l1 la misma lista que tiene asociada la variable l2, es decir, ambas variables apuntan a la misma dirección de memoria. Cualquier cambio que hagamos a través de l1 o l2 afectará a la misma lista.

Copia por valor l1 = list(l2): Crea una copia de la lista asociada a l2 en una dirección de memoria diferente y se la asocia a l1. Las variables apuntan a direcciones de memoria diferentes que contienen los mismos datos. Cualquier cambio que hagamos a través de l1 no afectará a la lista de l2 y viceversa.

```
>>> a = [1, 2, 3]
>>> # copia por referencia
>>> b = a
>>> b
[1, 2, 3]
>>> b.remove(2)
>>> b
[1, 3]
>>> a
[1, 3]
>>> a = [1, 2, 3]
>>> # copia por referencia
>>> b = list(a)
>>> b
[1, 2, 3]
>>> b.remove(2)
>>> b
[1, 3]
>>> a
[1, 2, 3]
```

Es posible anidar listas, listas multidimensionales, (crear listas que contengan otras listas), por ejemplo:

```
>>>
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

3.2 Tuplas

Colecciones de objetos que representan secuencias ordenadas de objetos de distintos tipos. A diferencia de las listas son inmutables, es decir, que no cambian durante la ejecución. Se representan mediante paréntesis y los elementos se separan por comas.

Ejemplo. (1, 'dos', 3)

```
»> tupla1 = ()
»> tupla2 = (“a”,1,True)
>>> tupla3=tuple()
>>> tuple4=tuple([1,2,3]) en este caso convertimos una lista en una tupla
```

En las tuplas se pueden realizar las siguientes operaciones:

- Las tuplas se pueden recorrer.
- Operadores de pertenencia: `in` y `not in`.
- Concatenación: +
- Repetición: *
- `Indexación`

Entre las funciones definidas podemos usar: len, max, min, sum, sorted
Entre los métodos : count

Podemos desempaquetar una tupla

mitupla=(“Ana”, 12 , 2 , 1999)
nombre,dia,mes,anyo=mitupla

Las tuplas son más rápidas porque ocupan menos espacio en memoria. Si queremos una lista en la que no variamos sus elementos usaremos tupla en lugar de lista.

Nos servirán también para utilizarlas como claves en los diccionarios.

3.3 Rangos

Es un tipo de secuencias que nos permite crear secuencias de números. Es un tipo inmutable y se suelen utilizar para realizar bucles.

Al crear un rango (secuencia de números) obtenemos un objeto que es de la clase range

Veamos algunos ejemplos, convirtiendo el rango en lista para ver la secuencia:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Entre las funciones definidas podemos usar: len,max, min, sum, sorted.

Ademas un objeto range posee tres atributos que nos almacenan el comienzo, final e intervalo del rango:

```
>>> rango = range(1,11,2)
>>> rango.start
1
>>> rango.stop
11
>>> rango.step
2
```

4.3 Conjuntos

Conjuntos (desordenados) de datos (a los que se puede calcular una función hash), en los que no existen repeticiones. Es un tipo de datos mutable.

Normalmente se usan para comprobar si existe un elemento en el conjunto, eliminar duplicados y cálculos matemáticos, como la intersección, unión, diferencia,...