Unidad 2: Tipos de datos
Listas
Instrucciones Iterativas

Objetivos

- Identificar las propiedades de las listas como herramienta de agrupación de datos.
- Aplicar indexación básica y slicing para la manipulación de datos representados con listas.
- Reconocer las funciones básicas de listas para el manejo de datos
- Implementar programas utilizando listas para la resolución de problemas

Propiedades de las Listas

Lista es un tipo de colección

- Una colección permite proner varios valores en una "variable"
- Una colección es muy útil porque podemos poner muchos valores agrupados convenientemente.
- Cualquier tipo de objeto puede estar en una lista

```
#una lista de frutas
frutas = ['uva', 'pera', 'durazno']

#una lista de nombres
amigos = ['jose', 'ana', 'elena']

#datos personales
info = ['Ana Maria', 'Armijos', 13.09, 20 ]
```

Crear una Lista

- Es posible crear una lista de constantes que se encierran entre corchetes y se separan por comas [,]
- Una lista puede tener elementos de distintos tipos de datos, incluido otra lista como elemento
- Una lista puede estar vacía [ ]

```
>>> print([1, 24, 76])
[1, 24, 76]

>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']

>>> print (['red', 24, 98.6])
['red', 24, 98.6]

>>> print ([ 1, [5, 6], 7])
[1, [5, 6], 7]

>>> print([])
[]
```

La función list()

- También es posible, crear una lista a partir de la función `list()`
- Junto con la función `range()`, devolverá una lista con la secuencia que genere range

```
>>> list()
[]

>>> list(range(5))
[0, 1, 2, 3, 4]

>>> list(range(1, 11, 2))
[1, 3, 5, 7, 9]

>>> list('Sofia')
['S', 'o', 'f', 'i', 'a']
```

¿Longitud de una Lista?

- La función `len()` toma una lista como parámetro y retorna el número de elementos de la lista.
- De hecho, `len()` nos indica el número de elementos de cualquier conjunto o secuencia (por ejemplo de una cadena...).

```
>>> cadena = 'Hello Bob'
>>> len(cadena)
9
>>> x = [ 1, 2, 'joe', 99 ]
>>> len(x)
4
```

Concatenación de Listas

- Podemos concatenar dos listas con el operador +, pero se devuelve una nueva lista.
- Se puede usar el operador +=, al igual que en los tipos de datos de cadenas y de enteros
- El operador * funciona como un repetidor, al igual que en las cadenas.

```
>>> a = [1, 2, 3]
>>> a = a + [6, 1]
>>> a
[1, 2, 3, 6, 1]
>>> a += [8, 9]
>>> a
[1, 2, 3, 6, 1, 8, 9]
>>> li = [1, 2] * 3
>>> li
[1, 2, 1, 2, 1, 2]
```

Indexación básica y slicing

Indexación en listas

- Podemos acceder a cualquier elemento de una lista a partir del nombre de ésta y el índice del elemento entre corchetes.
- Cada elemento tiene un tipo de dato y por lo tanto sus propias operaciones permitidas

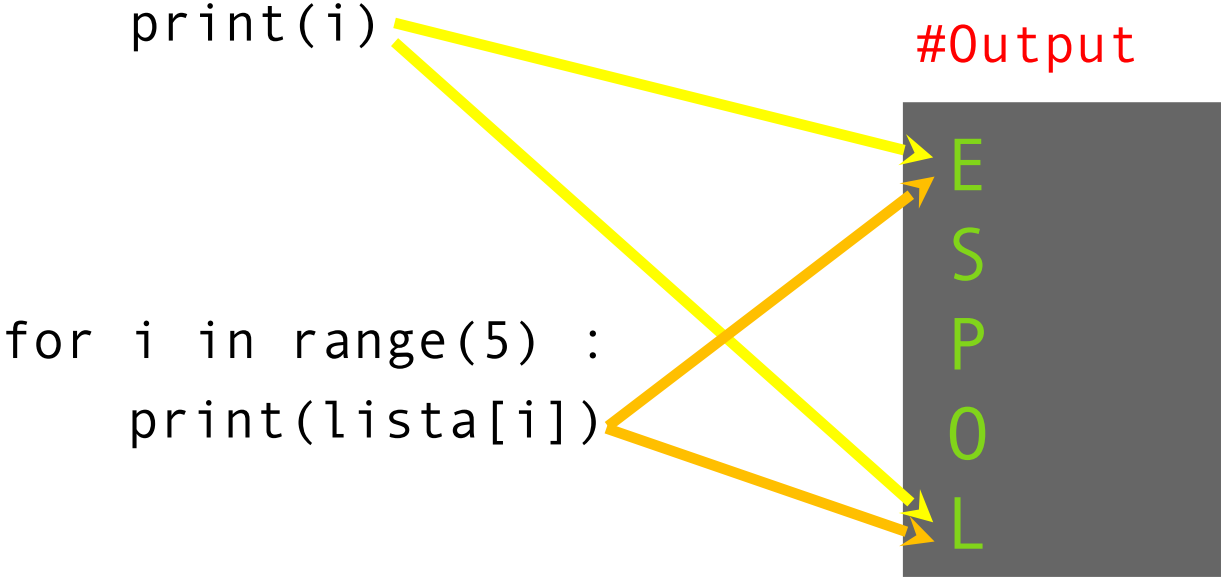

```
>>> hijos = [ 'Joshua', 'Sophia', 3 ]
>>> hijos[1]
'Sophia'
>>> hijos[0] * hijos[2]
'JoshuaJoshuaJoshua'
>>> hijos[0] + ' ' + hijos[1]
'Joshua Sophia'
```

Iterar sobre una lista

- La instrucción `for` nos permite iterar sobre los elementos de una lista.
- En ocasiones es necesario iterar mediante los índices de los elementos

```
#Programa
lista = ['E', 'S', 'P', 'O', 'L']
for i in lista :
    print(i)

for i in range(5) :
    print(lista[i])
```



Indices negativos

- Un índice negativo da acceso a los elementos del final de la lista, contando hacia atrás. El último elemento de cualquier lista no vacía es siempre `li[-1]`
- Si los índices negativos le resultan confusos, piénselo de este modo: `li[n] == li[n - len(li)]`.
- De modo que en esta lista: `li[2] == li[ 2 - 5]`

```
>>>li
['a','b','mil','z','exa']

>>>li[-1]
'exa'

>>>li[-3]
'mil'
```

Se pueden cortar listas con slicing:

```
>>> t = [9, 10, 8, 3, 7, 15]
>>> t[1:3]
[10, 8]
>>> t[:4]
[9, 10, 8, 3]
>>> t[3:]
[3, 7, 15]
>>> t[:]
[9, 10, 8, 3, 7, 15]
```

Advertencia: Como en las cadenas, el segundo número es "hasta pero no inclusive".

Funciones básicas

Añadir elementos

- `.append(ele)` añade un elemento al final de la lista.
- `.insert(pos, ele)` inserta un elemento en el índice del elemento que será desplazado de su posición (pos)
- `.extend(li)` concatena, usa un único argumento: una lista. A diferencia del operador + no se crea una nueva lista

```
>>> li
['a', 'z', 'ex']
>>> li.append("new")
>>> li
['a', 'z', 'ex', 'new']
>>> li.insert(2, "new")
>>> li
['a', 'z', 'new', 'ex', 'new']
>>> li.extend(["two"])
>>> li
['a','z','new','ex','new','two']
```

Búsqueda de elementos

- `.index(ele)` encuentra la primera aparición de un valor en la lista y devuelve su índice.
- Si el valor no se encuentra en la lista, Python lanza una excepción.
- Para comprobar si un valor está en la lista, utilice antes `in`:

```
if 'c' in li :
   pos = li.index('c')
```

```
>>> li
['a', 'z', 'new', 'ex', 'new', 'two']
>>> li.index("ex")
3
>>> li.index("new")
2
>>> li.index("c")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'c' is not in list
>>> "c" in li
False
```

Eliminar elementos

`.remove()` elimina la primera aparición de un valor en una lista. Si el valor no se encuentra en la lista, Python lanza una excepción.

`.pop()` elimina el último elemento de la lista, y devuelve el valor que ha eliminado. Es diferente de `li[-1]`, que devuelve el valor sin modificar la lista, y de `li.remove(valor)`, que modifica la lista sin devolver ningún valor.

```
>>> li
['a', 'z', 'new', 'ex', 'new', 'two']
>>> li.remove("z")
>>> li
['a', 'new', 'ex', 'new', 'two']
>>> li.remove("new")
>>> li
['a', 'ex', 'new',  'two']
>>> li.remove("c")
ValueError: list.remove(x): x not in  list
>>> x = li.pop()
>>> print(x, li)
'two' ['a', 'ex', 'new']
```

Dividir cadenas `str.split()`

- La función .split(sepador) divide una cadena en partes de acuerdo a la posición de una sub-cadena separador
- La función retorna una lista de cadenas.
- .split() sin argumentos dividirá la cadena por el espacio en blanco
- Usualmente, se utiliza .split() y unpacking para asignar a más de una variable, solo sí, el número de elementos de la lista resultado es conocido

```
>>> cad = 'Python is fun'
>>> cad.split(' ')
['Python', 'is', 'fun']
>>> mesj = '2022-05-31'
>>> mesj.split('-')
['2022', '05', '31']
>>> l = mesj.split('-')
>>> y, m, d = mesj.split('-')
>>> print( y, m, d)
2022 05 31
```

Juntar cadenas `str.join(list)`

- La función .join() es una función de las usada para concatenar una lista de cadenas en una sola cadena con un separador específico.
- Generalmente usada para obtener una nueva cadena a partir de las cadenas de la lista
- No añade el separador al final de la cadena

```
>>> pals =
['Python', 'is', 'fun']
>>> mesj = '-'.join(pals)
>>> mesj
'Python-is-fun'
>>> nums =
['2024', '12', '31']
>>> mesj = ' '.join(nums)
>>> mesj
'2024 12 31'
```