Listas, Tuplas y Diccionarios

IITA 2023

Primero…un repaso:

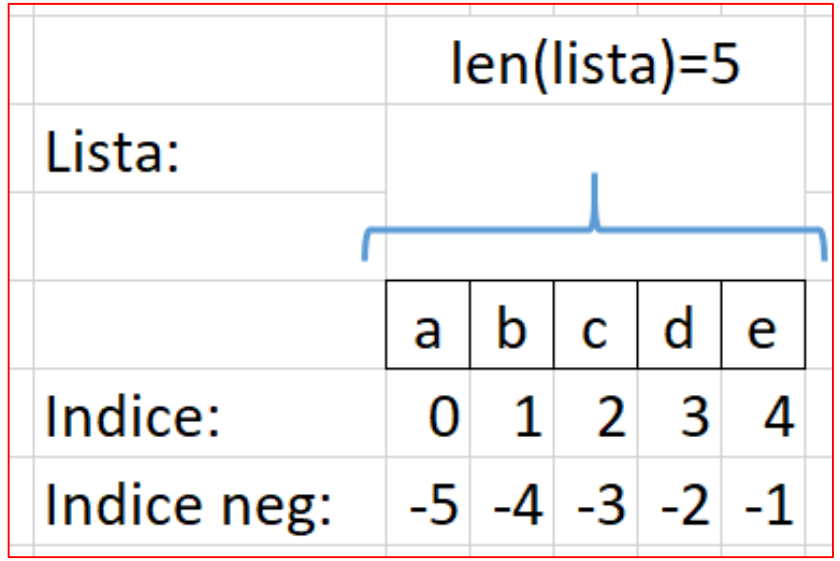
- Python…¿Qué es?
- SOFTWARE
- HARDWARE
- LENGUAJE DE PROGRAMACIÓN
- PROGRAMA
- TIPO DE DATO
- VARIABLE

- Estructuras:
  - Selectivas
    - Simples
    - Múltiples
  - Repetitivas

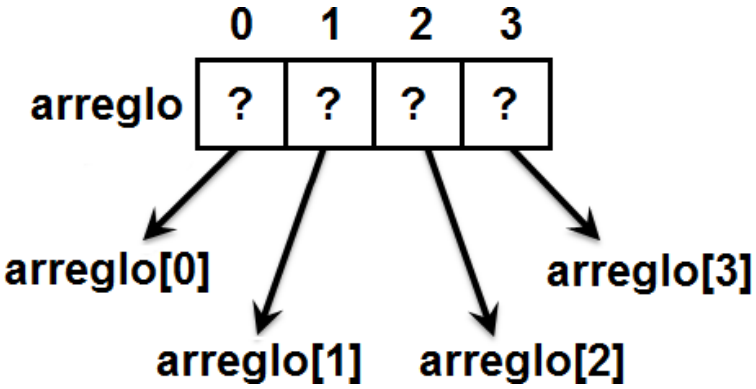
Listas []

- Se define usando corchetes, es una lista de elementos a los cuales podemos acceder mediante un índice que indica su posición en la lista, empezando siempre por el número 0.
- Ejemplo:

```
        lista_ejemplo = [“a”, “b”, “c”, “d”, “e”]
        print(lista_ejemplo)
>>> [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
        print(lista_ejemplo[2])
>>> ‘c’
        print(lista_ejemplo[-1])
        >>> ‘e’
```



Listas []



- **"Porción de lista":** se puede obtener un subconjunto de una lista, especificando dos índices
  - →El valor de retorno es una nueva lista, que contiene todos los elementos de la lista original, en orden, empezando con el primer índice de la porción, hasta el segundo índice de la porción (sin incluir a éste):

```
lista_ejemplo = ["a", "b", "c", "d", "e"]
print(lista_ejemplo[2:4])
>>> ['c', 'd']
```

Listas []

Operaciones

lista_ejemplo = [“a”, “b”, “c”, “d”, “e”]

→**Append():** añade un elemento <u>al final</u> de la lista

lista_ejemplo.append(‘K’)

>>> [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘K’]

→**Insert():** inserta un elemento en una lista, <u>especificando por índice</u> la posición del mismo

lista_ejemplo.insert(2, ‘ ’)

>>> [“a”, “b”, “Y”, “c”, “d”, “e”]

→**Extend():** concatena listas. Debe llamarse con un <u>único argumento</u>: otra lista!

lista_ejemplo.extend([“casa”, “edificio”])

>>> [“a”, “b”, “c”, “d”, “e” , “casa”, “edificio”]

Listas []

Operaciones

lista_ejemplo = [“a”, “b”, “c”, “d”, “e”, “b”]

→**remove():** elimina la primera aparición de un valor en la lista

lista_ejemplo.remove(“b”)

>>> lista_ejemplo = [“a”, “c”, “d”, “e”, “b”]

→ **pop():** remueve el ultimo elemento existente en la lista

lista_ejemplo.pop()

>>> [“a”, “b”, “c”, “d”, “e”]

→ Operaciones sobre listas: en python podemos operar directamente con una lista

lista_ejemplo+=[“casa”, “perro”]

>>> [“a”, “b”, “c”, “d”, “e”, “b”, “casa”, “perro”]

li = [1, 2] * 3

print(li)

>>>[1, 2, 1, 2, 1, 2]

→ “Compresión de listas”

A practicar un poco...

1. Meter los números del 1 al 35 en una lista y mostrarla en pantalla. Hacer lo mismo para un rango de números indicado por un usuario.

2. Pide una cadena (string) por teclado, mete los caracteres en una lista sin espacios. (***)

3. Realizar un programa que inicialice una lista con 10 valores aleatorios (del 1 al 10) y posteriormente muestre en pantalla cada elemento de la lista junto con su cuadrado y su cubo.

A practicar un poco...

4. Pedir una palabra al usuario, meter los todos sus caracteres en una lista y mostrarla en pantalla.

5. Realizar un programa que inicialice una lista con 10 números ingresados por el usuario y luego muestre en pantalla cual elemento es el menor

Tuplas ()

- Se define usando paréntesis. Es una lista **inmutable**, no puede modificarse de ningún modo después de una creación
- Las Tuplas son más rápidas que las listas. Por ende → si trabajamos con un conjunto de valores CONSTANTES y lo único que hacemos es recorrer dicho conjunto, conviene utilizar Tuplas antes que listas

```
>>> t = ("a", "b", "mpilgrim", "z", "example")
>>> t
('a', 'b', 'mpilgrim', 'z', 'example')
>>> t[0]
'a'
>>> t[-1]
'example'
>>> t[1:3]
('b', 'mpilgrim')
```

- **Las Tuplas NO TIENEN métodos asociados!!!**

Listas y Tuplas ()

- OBSERVACIÓN: las Tuplas pueden convertirse en listas, y viceversa.

La función incorporada *tuple()* toma una lista y devuelve una tupla con los mismos elementos

```
lista_ej=[2,3,4,5,6]
tupla_ej=tuple(lista_ej)
print(tupla_ej)
```

→

La función incorporada *list()* toma una tupla y devuelve una lista con los mismos elementos

```
tupla_ej=(2,3,4,5,6)
lista_ej=list(tupla_ej)
print(lista_ej)
```

→

A practicar un poco...

1. Crea una tupla con números, pide un numero por teclado e indica cuantas veces se repite.

2. Crea una tupla con valores ya predefinidos del 1 al 10, pide un índice por teclado y muestra los valores de la tupla.

A practicar un poco...

3. Crea una tupla que contenga todos los meses de un año. Luego pedir al usuario un numero y mostrar que mes representa

Diccionarios {‘key’: ‘value’}

- Define una relación uno a uno entre claves y valores. Por ejemplo

```
dicc_ejemplo = {‘usuario’: ‘nmartinelli’, ‘contraseña’: ‘1234’}

print(dicc_ejemplo[‘usuario’])

>>> ‘nmartinelli’
```

→Se pueden obtener los valores por su clave, pero no se pueden obtener las claves por el valor

→Para modificarlos, accedemos por la clave de la siguiente manera:

```
dicc_ejemplo[“usuario”]=“nmartinelli23”
dicc_ejemplo[“telefono”]=“155555555”
print(dicc_ejemplo)
>>> {‘usuario’: ‘nmartinelli23’, ‘contraseña’: ‘1234’, ‘telefono’: ‘155555555’}
```

Diccionarios {'key': 'value'}

Observaciones

- No pueden haber claves duplicadas
- Se pueden añadir nuevos pares clave-valor en cualquier momento (es dinámico)
- Los diccionarios no tienen concepto de orden entre sus elementos, se basan en las claves directamente
- Miren los valores!

```
diccionario = {'nombre': 'Carlos', 'edad': 22, 'cursos':['Python','Django']}
```

A practicar un poco...

1. Crea un programa que pida al usuario el nombre de un mes (Enero, Febrero, etc.) y diga cuántos días tiene (por ejemplo, 30. Para simplificarlo vamos a suponer que febrero tiene 28 días. Usar diccionarios

2. Crear un programa que pida al usuario un nombre de un alumno, y luego muestre la lista de notas de ese alumno. Usar diccionarios