TC1028

Pensamiento Computacional para Ingeniería

Listas o arreglos

Tecnológico de Monterrey

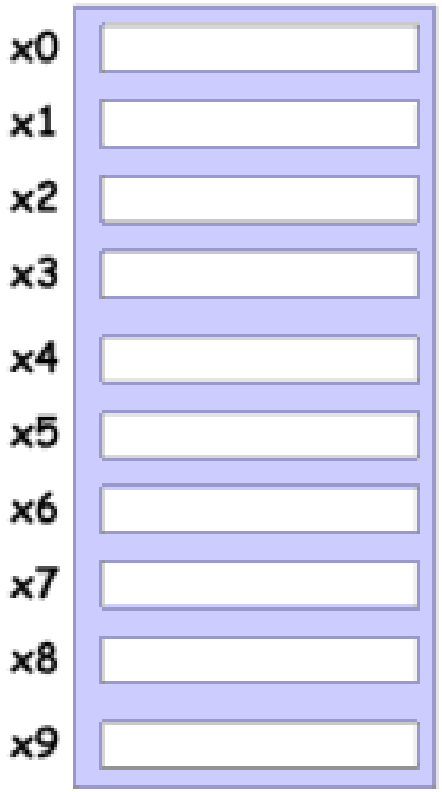
Listas o arreglos

- En sesiones pasadas definimos a la **variable** como la asociación entre **un nombre** y **una localidad.**
- Hasta ahora, si necesitamos guardar varios valores necesitamos definir una variable para cada uno de ellos.

Listas o arreglos

Por ejemplo, supongamos que necesitamos trabajar con 10 valores enteros; el código para definir a estas 10 variables sería algo como lo siguiente:

x0
x1
x2
x3
x4
x5
x6
x7
x8
x9



Listas o arreglos

¿Qué tal si ahora necesitamos definir un procedimiento en donde se asigne a cada variable el valor **-1**?, el código sería algo como lo siguiente:

```
def iniciaMenosUno ( ):
    x0=-1
    x1=-1
    x2=-1
    x3=-1
    x4=-1
    x5=-1
    x6=-1
    x7=-1
    x8=-1
    x9=-1
```

Listas o arreglos

- Pero ¿Qué tal si en lugar de trabajar con **10** necesitamos definir **1000**?

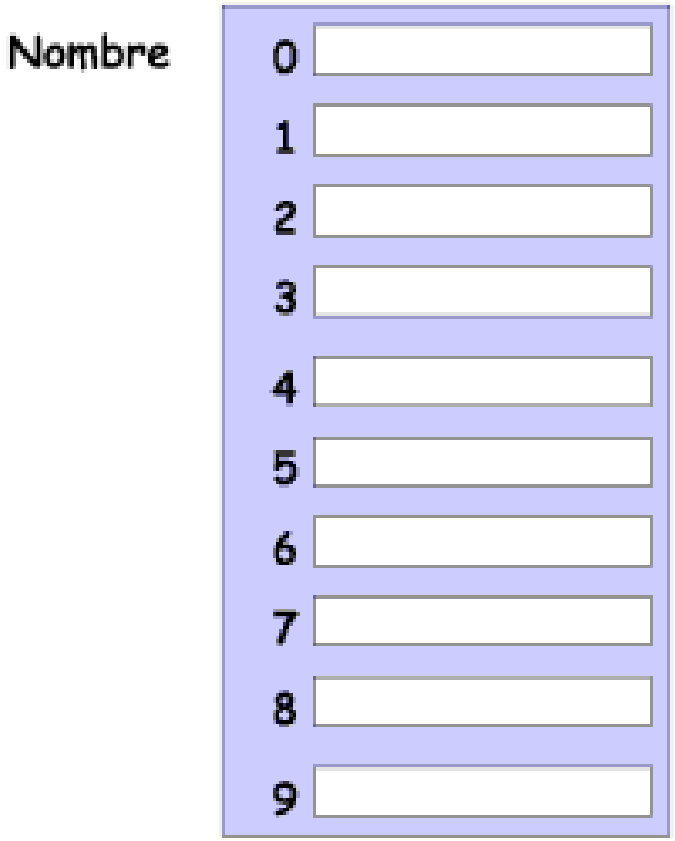
- Que tal si en vez de trabajar con varias variables aisladas, trabajamos con **un solo contenedor de variables**.

Listas o arreglos

- Una variable **arreglo** es una asociación entre **un nombre** y **un grupo de localidades.**
- Cada localidad está asociada con un número, de tal manera que para identificar a localidad específica del arreglo es necesario escribir su número.
- La **primera** localidad siempre corresponde a la localidad **0** y la **última** corresponde a **n-1**, en donde **n** es el número de localidades del arreglo.

Listas o arreglos

Por ejemplo, en un arreglo de **10** localidades, la primera localidad es **0** y la última localidad es **9**.



Listas o arreglos

La característica de una variable arreglo son los corchetes(**[ ]**). Una definición de variables que los incluya, indica que la variable es un **arreglo.**

lista = []

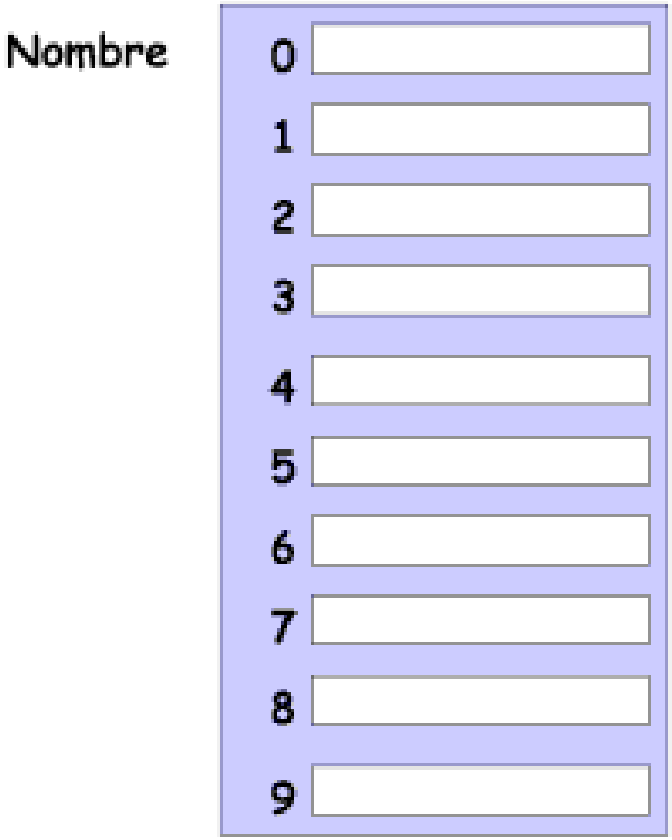
lista = ['a', 1 , "Buen dia"]

Listas o arreglos

- Una vez que hemos declarado la variable arreglo, ¿Cómo tenemos acceso a los valores? Debemos recordar que las localidades de un arreglo están numeradas de **0** a **n-1** (en donde **n** es el tamaño del arreglo).
- Para hacer referencia a una localidad específica del arreglo debemos escribir el **nombre de la variable y entre los corchetes el número de la localidad.**

Listas o arreglos

- La manera como indicamos a la primera localidad del arreglo A es A[**0**] y la última localidad es A[**9**].



Listas o arreglos

- La forma de asignar un valor a una localidad específica del arreglo es la siguiente:

nombre[localidad] = valor

- En donde **nombre** es el nombre de la variable arreglo, **localidad** es el número de la localidad del arreglo (entre **0** y **n-1**, donde **n** es el tamaño del arreglo) y **valor** es cualquier dato del tipo con que fue definido el arreglo.

Actividad Grupal

Declarar un arreglo llamado **numeros** y guardar en cada localidad los valores 0.2, 0.4, 0.6 y 0.8 respectivamente

Solución!!!

numeros = [0.2, 0.4, 0.6, 0.8]

Actividad Grupal

Sumar el valor de **8** a cada una de las localidades del arreglo del ejemplo anterior

Solución!!!

numeros[0] = numeros[0]+ **8**

numeros[1]= numeros[1]+ **8**

numeros[2]= numeros[2]+ **8**

numeros[3]= numeros[3]+ **8**

Listas o arreglos

- La estructura compañera de los arreglos es el ciclo **for**.
- El acceso a los elementos de un arreglo puede ser de dos formas:

1. **A través del iterador**

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

2. **A través del índice**

```
for i in range(0,len(lista)):
        print(lista[i])
```

La función **len** obtiene el número de elementos en la lista.

Actividad Grupal

Define un arreglo que almacene **10**

números enteros

Solución!!!

La declaración de la variable arreglo **A** que contiene **10** valores **enteros** es la siguiente:

A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Actividad Grupal

Escribe el ciclo para imprimir la lista anterior en la pantalla.

Solución!!!

El siguiente código imprime la lista en la pantalla:

```
for ele in A:

    print(ele)
```

listas.py

```
1  A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  for ele in A:
4      print(ele)
```

Shell

```
>>> %Run listas.py

  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
```

Actividad Grupal

Escribe el ciclo para imprimir la lista en la pantalla.

Usa **end = ' '** para evitar el salto de línea por default así:

print(ele, end = ' ')

Solución!!!

El siguiente código imprime la lista en la pantalla, sin salto de línea.

```
for ele in A:
    print(ele, end='  ')
```

listas.py

```
1  A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  for ele in A:
4      print("%d "% ele, end='')
```

Shell

```
>>> %Run listas.py
  1 2 3 4 5 6 7 8 9 10
```

Actividad Grupal

Establece el valor de la localidad 5 a 35

Solución!!!

El valor de la localidad **5** se modifica a **35**.

A [5] = 35

```
1  A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  A[5] = 35
4  for ele in A:
5      print(ele, end=' ')
```

Shell

```
>>> %Run listas7.py
  1 2 3 4 5 35 7 8 9 10
```

Actividad Grupal

Establece el valor de la localidad 7 a la suma de la localidad 6 y la 9.

Solución!!!

El valor de la localidad **7** se modifica con la suma de la localidad **6** y la localidad **9**.

A [7] = A[6] + A[9]

```
1  A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  A[5] = 35
4  A[7] = A[6] + A[ 9 ]
5  for ele in A:
6      print(ele, end=' ')
```

Shell

```
>>> %Run listas7.py
  1 2 3 4 5 35 7 17 9 10
```

Actividad Grupal

Establece el valor de la localidad 8 a 3 veces el valor de la localidad 4, menos 57.

Solución!!!

Se establece el valor de la localidad 8 a 3 veces el valor de la localidad 4, menos 57.

A [8] = A[4]*3 - 57

```
1  A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  A[5] = 35
4  A[7] = A[6] + A[ 9 ]
5  A[8] = A[4]*3 - 57
6  for ele in A:
7      print(ele, end=' ')
```

Shell

```
>>> %Run listas.py
  1 2 3 4 5 35 7 17 -42 10
```

Actividad Grupal

¿Qué imprime el siguiente código?

```
numeros = [0,1,2,3,4]
acum = 0
for num in numeros:
        if( num % 2 == 0 ):
                acum = acum + num * 5
print(acum)
```

Listas o arreglos

El acceso a los elementos de un arreglo puede ser de dos formas:

1. **A través del iterador**

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

2. **A través del índice**

```
for i in range(0,len(lista):
    print(lista[i])
```

La función len obtiene el número de elementos en la lista

```
lista=[8,5,2]
print(len(lista))
```

```
1  lista=[8,5,2]
2  print(len(lista))
3
```

Shell ×

```
>>> %Run listas8.py
3
```

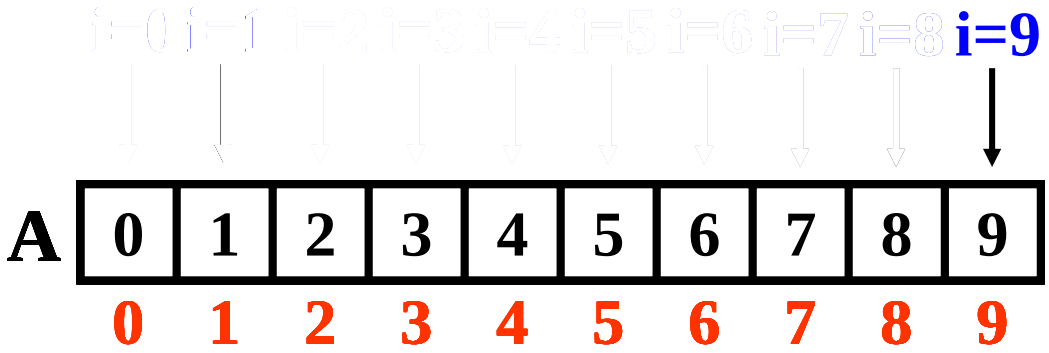
Actividad Grupal

Analiza las siguientes simulaciones para comprender la relación del ciclo **for** con los arreglos.

Actividad Grupal

Inicializa un arreglo de **10** valores con su número de localidad:

A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

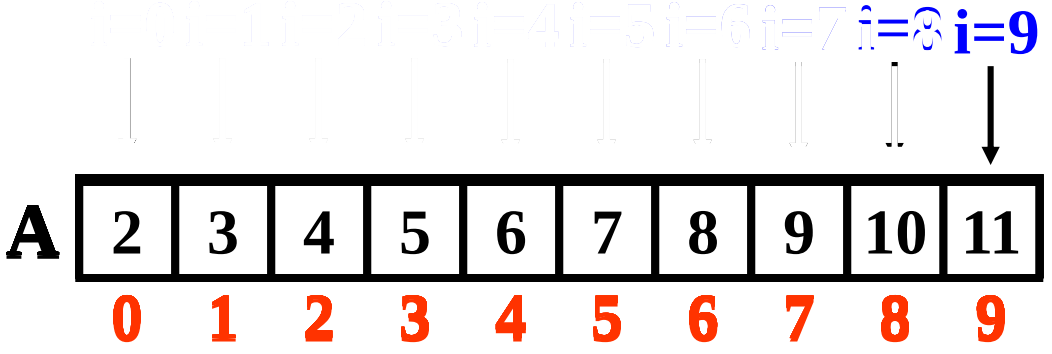


Actividad Grupal

A continuación mostramos la función **sumaDos**, que recibe un arreglo de **10** valores enteros y le suma un dos (**2**) a cada una de las localidades del arreglo.

```
def sumaDos (B):
    for i in range(len(B)):
            B[i] = B[i] + 2
```

```
A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sumaDos(A)
```



Actividad Grupal

Escriba la función **imprimeArreglo**, que recibe un arreglo y despliega en pantalla el contenido del arreglo.

```
def imprimeArreglo(B):
    for i in range(len(B)):
        print("Arreglo[", i,"] = ", B[i])
```

Actividad Grupal

Integrar todas las funciones en un solo programa

para verificar su funcionamiento

Actividad Grupal

```
def sumaDos (B):
    for i in range(len(B)):
        B[i] = B[i] + 2

def imprime(B):
    for i in range(len(B)):
        print("Arreglo[", i, "] = ", B[i])

A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
imprime(A)
sumaDos(A)
imprime(A)
```

Funciones y métodos de listas

Descripción	FUNCIONES	MÉTODOS	Ejemplo
Obtiene el número de elementos en la lista	**len**(lista)		**lista=[8,5,2]** **len(lista)** regresa 3
Agrega el valor indicado en el parámetro al final de la lista		lista.**append**(valor)	**lista.append(10)** La nueva lista sería: [8, 5, 2, **10**]
Inserta el valor indicado en la posición indicada		lista.**insert**(posición, valor)	**lista.insert(1, 4)** La nueva lista sería: [8, **4**, 5, 2, 10]
Borra el elemento indicado de la lista en la posición indicada	**del**(lista[indice])		**del(lista[1])** Borarría el 4. [8, 5, 2, 10]
Borra el elemento que tenga el valor que se recibe como parámetro		lista.**remove**(valor)	**lista.remove(10)** Borraría el valor 10 de la lista [8, 5, 2]
Obtiene el valor que se encuentra en la posición indicada por el índice y lo borra de la lista		lista**.pop**(indice)	**valor = lista.pop(1)** valor obtendría el valor de 5 y lo borraría de la lista. [8, 2]
Regresa la lista ordenada, sin modificar la lista original.	**sorted**(lista)		**lista1=['j','a','g','b']** **lista2=sorted(lista1)** **print(lista1)** Imprimirá ['j','a','g','b'] **print(lista2)** Imprimirá ['a', 'b', 'g', 'j']
Ordena los elementos de la lista de mayor a menor. Si se indica el parámetro reverse en True se ordena de mayor a menor		lista**.sort**(**[reverse=True]**)	**lista1=['j','a','g','b']** **lista1.sort()** **print(lista1)** Imprimirá ['a', 'b', 'g', 'j']