



UNIVERSIDAD NACIONAL
DE ASUNCIÓN
**FACULTAD DE
INGENIERÍA**



Estructuras de datos (Python)

Cátedra de Fundamentos de Programación




















Tipos de datos en Python



¿Qué veremos hoy?

- Necesidad de contar con estructuras de datos
- Vectores (arreglos unidimensionales)
 - Listas en Python
 - Acceso a elementos
 - Lectura
 - Impresión
- Otras estructuras:
 - Tuplas
 - Diccionesarios
 - Conjuntos
- Ejercicios

Python List Methods

Input	Method	Output
	<code>.append()</code>	
	<code>.insert(1, )</code>	
	<code>.pop(1)</code>	
	<code>.remove()</code>	
	<code>.reverse()</code>	
	<code>.sort()</code>	
	<code>.index()</code>	2
	<code>.count()</code>	2

Necesidad de estructuras de datos

Las variables simples únicamente pueden almacenar un dato. En ejercicios anteriores, hemos trabajado con listas de números (calificaciones de alumnos de una misma clase, trabajadores de una empresa, etc.); y su manejo con variables simples puede ser poco práctico, además de no considerar el almacenamiento de las entradas.

```
a = 5 #int  
d = 3.2 #float  
l = "ab" #string
```

Pregunta 1: ¿Cómo podríamos ordenar N enteros utilizando variables simples?

$\{5, 1, 8, 7, 3, 4, \dots\}$

Pregunta 2: ¿Cómo calcularíamos la cantidad de alumnos que obtuvieron una nota inferior al promedio del curso en cierta materia?

Estructuras de datos

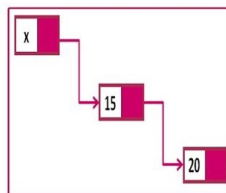
Para salvar estas situaciones, la mayoría de los lenguajes de programación incluyen **estructuras de datos**. Una estructura de datos es una colección de datos que pueden ser caracterizados por su organización y las operaciones que se definen en ella (*acceso, inserción, borrado, etc*).

Las estructuras de datos se dividen en:

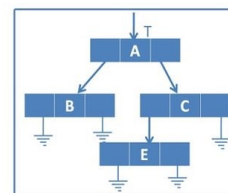
- **Estáticas** (tamaño definido de antemano): arreglos, registros, cadenas
- **Dinámicas** (no tiene limitaciones de tamaño): listas enlazadas, pilas, colas, árboles, grafos.



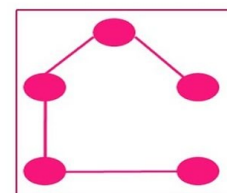
Sorting



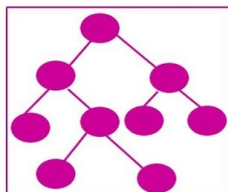
Link list



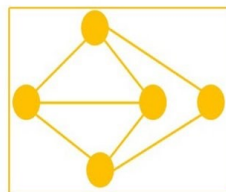
list



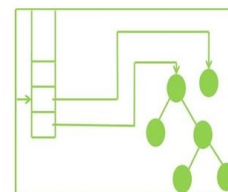
spanning tree



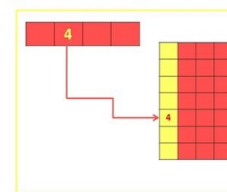
Tree



Graph



Stack



Hashing

Arreglos


La **estructura de datos básica** que soporta la mayoría de los lenguajes son los **arreglos** (*arrays*), siendo el **vector** un arreglo de una dimensión, y la **matriz** de dos dimensiones.

Un **arreglo** es una secuencia de posiciones de la memoria central a las que se puede acceder directamente, que contiene **datos del mismo tipo** y pueden ser seleccionados individualmente mediante el uso de *índices*.

En Python no existe el arreglo de forma nativa, por lo que emplearemos “listas” para representarlos.

Ejemplo de vector: nota (notas de una clase de n alumnos)

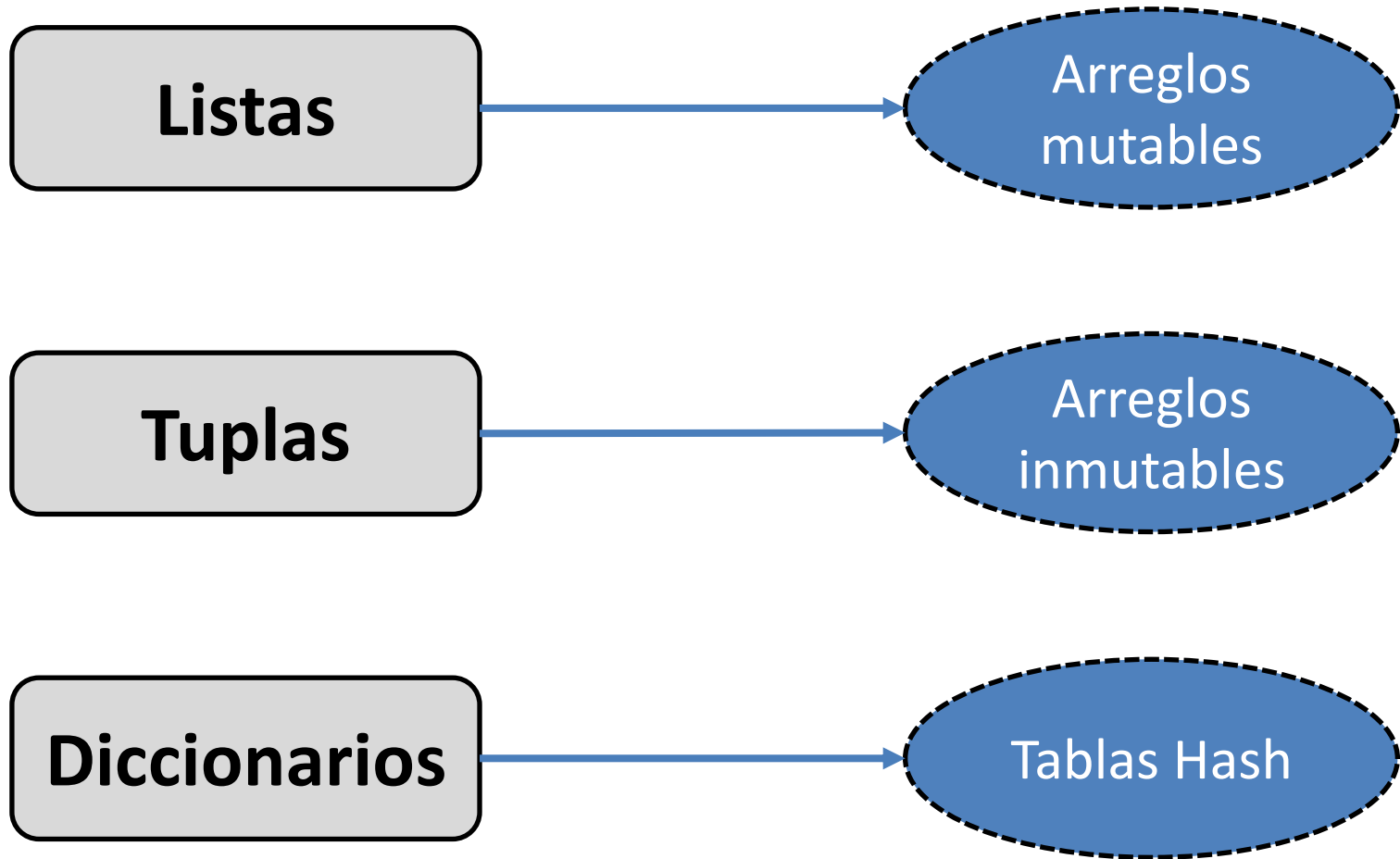
nota 0	nota 1	nota 2	...	nota i	...	nota $n-1$
--------	--------	--------	-----	----------	-----	------------



notas[2]

Observación importante: en Python, el primer elemento se representa por el índice 0. Si el tamaño del vector (*lista*) es n , entonces el último elemento tiene un índice $n-1$.

Estructuras básicas en Python



Listas en Python

Una lista es un tipo dato estructurado de tipo mutable (**puede modificarse sus elementos**). A diferencia de un arreglo, puede contener cualquier tipo de elemento. Su definición es simplemente con corchetes.

```
A = []           # define una lista vacía
N = [10,20,30]   # define una lista de elementos numéricos
S = ['hola','que','tal'] # define una lista de cadenas
B = [True, False, True] # define una lista de booleanos
M = [10,'hola',True]  # define una lista de elementos variados
```

La *lista* **nota** tiene subíndices o índices de sus elementos (0, 1, 2, ..., i , ..., $n-1$) que indican la posición de un elemento particular dentro del arreglo (de tamaño n). Por ejemplo, si se desea modificar el tercer elemento de una lista:

nota[2] = 80

Nombre de la lista Posición del elemento

Operaciones sobre listas

Las listas en Python son **dinámicas**: pueden crecer o decrecer sin redefinirse.

```
>>> A = [10,20,30] # define una lista de elementos numéricos
>>> A[0] = 40      # cambia el elemento de la posición 0
>>> A
[40, 20, 30]
>>> len(A)         # retorna la cantidad de elementos de la lista
3
>>> A.append(60)    # Agrega el elemento 60 al final
>>> A
[40, 20, 30, 60]
>>> A[-1]          # se accede al último elemento (índice negativo)
60
```

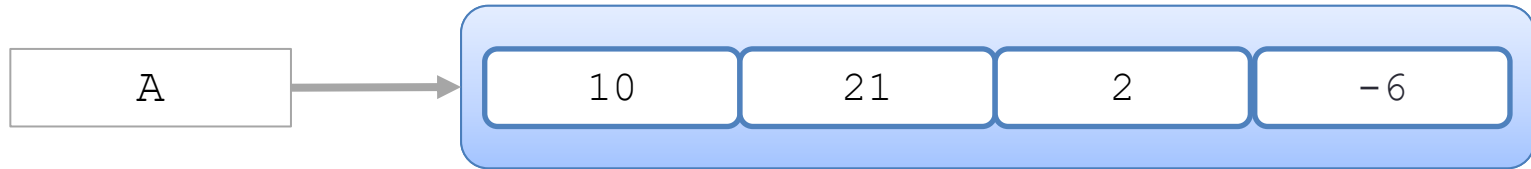
Observación importante: las listas y tuplas son **secuencias** en Python, ya que usan como índices a números enteros para acceder a sus elementos.

Operaciones sobre listas

Algunas funciones asociadas a listas son las siguientes (<lista> es el nombre de la lista):

Función	Descripción
<lista>.append(<valor>)	Agrega un nuevo elemento al final de la lista
<lista>.count(<valor>)	Retorna el número de ocurrencias de un valor en la lista
<lista>.extend(<var>)	Extiende la lista a partir de otro elemento estructurado (una lista por ejemplo)
<lista>.index(<valor>)	Retorna el índice de la primera ocurrencia del elemento <valor>
<lista>.insert(<ind>,<valor>)	Inserta un nuevo elemento antes de la posición indicada
del <lista>[<pos>]	Elimina un elemento en cierta posición <pos>
<lista>.remove(<valor>)	Remover la primera ocurrencia en el vector del elemento <valor>
len(<lista>)	Retorna la cantidad de elementos del arreglo
<lista>.clear()	Elimina todos los elementos de la lista

Obtener una parte de una lista



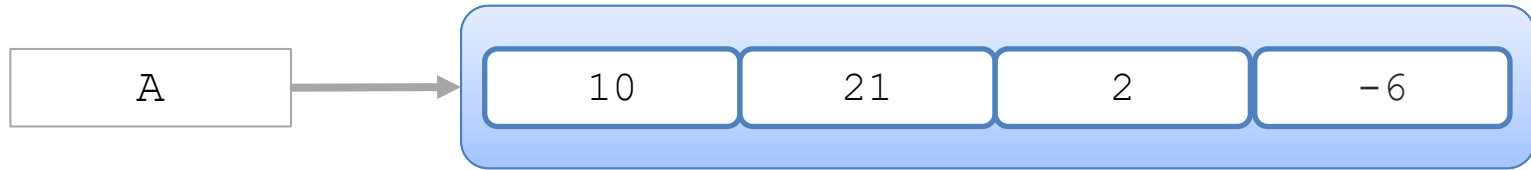
Se puede obtener una porción de una lista (que crea otra lista):

```
>>> A[1:3] #elementos desde 1 hasta 3-1
[21, 2]
>>> A[:3] #elementos desde el comienzo hasta 3-1
[10, 21, 2]
>>> A[1:] #elementos desde 1 hasta el final
[21, 2, -6]
```

Entonces, para copiar una lista:

```
>>> B = A #A y B son la misma lista!
>>> B = A[:] #A y B son dos listas distintas
>>> B
[10, 21, 2, -6]
```

Obtener una parte de una lista



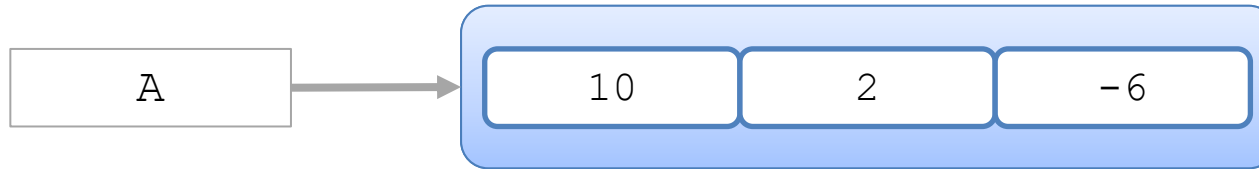
Las listas pueden concatenarse:

```
>>> A + [1,4]
[10, 21, 2, -6, 1, 4]
>>> A #Lo anterior no cambia la lista A
[10, 21, 2, -6]
```

La función **del** puede emplearse para eliminar elementos:

```
>>> del A[1] #elimina el elemento en la posición 1
>>> A
[10, 2, -6]
```

Obtener una parte de una lista



Las listas pueden extenderse:

```
>>> A.extend([1,4])
>>> A
[10, 2, -6, 1, 4]
```

Puede contarse las apariciones de cierto elemento:

```
>>> B = [1,3,7,3,3,1,3,5]
>>> B.count(3) #Cuenta las apariciones de 3
4
```

Operadores de membresía

Operador	Descripción
in	True si encuentra una variable en la secuencia especificada, y False en caso contrario.
not in	True si NO encuentra una variable en la secuencia especificada, y False en caso contrario.

```
>>> A=[1,2,5,6]
>>> 3 in A
False
>>> 2 in A
True
```

Rango de índices

El número de elementos de una lista determina el rango de valores que puede tomar un índice. Así, si una lista tiene **N** elementos, el rango está determinado por **[0, N-1]**. Si se sobrepasa el valor máximo, se tendrá un error:

```
>>> A = [6,7,1,0]
>>> A[4] #Intento de acceso a un elemento inexistente
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    V[4]
IndexError: list index out of range
```

Listas – Lectura e impresión

```
#Se lee la cantidad de elementos
N = int(input("Ingrese la cantidad de elementos: "))
#Se leen los elementos de la lista
A = [] #Lista vacía de nombre A
for i in range(N):
    x = int(input(f"Ingrese A[{i}]: "))
    A.append(x) #Agregamos x al final de la lista
print("El tamaño de la lista es:", len(A))
#Impresión de la lista - Forma 1
print("\nLa lista es (1):")
print(A)
#Impresión de la lista - Forma 2
print("\nLa lista es (2):")
for i in range(N):
    print(f"A[{i}] = {A[i]}")
```

El ciclo **for**

El ciclo **for** en Python itera sobre un conjunto de elementos de alguna *secuencia* (lista, tupla, cadena), en el orden en que aparecen en la secuencia. Hasta ahora solo empleamos **range()**, que justamente genera una secuencia de números enteros.

Sin embargo, podemos iterar directamente sobre los elementos:

```
>>> B = [4,1,5,6,10]
>>> for x in B: #En cada iteración, x toma un elemento de B
...     print(x)
...
4
1
5
6
10
```


Cadenas o Texto(str)

- Es un tipo secuencial en Python (elementos accesibles por un índice)
- Es inmutable y está delimitado por comillas simples o dobles
- Cada elemento es de tipo UniCode (codificación de caracteres)
- Son comparables.

x = "Juan Perez"

y = 'Juan Perez'

z = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""

z = " La Facultad de Ingeniería es la "mejor "
facultad "

z = " La Facultad de Ingeniería es la \"mejor \"
facultad "

x*5 # que imprimiría? 😊 tema de examen

Caracteres

ord(a)

97

chr(65)

A

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.overcoded.net

```
print('\u0061\u0062\u0063')
```

<https://home.unicode.org/>

"\N{GREEK CAPITAL LETTER DELTA}" # También se puede usar el nombre del caracter



Cadenas o Texto(str)

Es posible acceder a porciones de la cadena, de la forma

`<cadena>[<inicio>, <fin>, <paso>]` donde `<fin>` y `<paso>` son opcionales.

```
x= " computación"  
print(x[0])
```

```
print(x[0:3])
```

```
print(x[3::2])
```

```
# pasar a mayúsculas  
a = "Hello, World!"  
print(a.upper())
```

```
# pasar a minúsculas  
  
a = "Hello, World!"  
print(a.lower())
```

```
# remover espacios  
a = " Hello, World! "  
print(a.strip())
```

```
# formato  
edad = 36  
txt = "Tengo {} años"  
print(txt.format(edad))
```

```
# formato  
cantidad = 3  
item = 567  
precio = 49.95  
orden = "quiero {} de {} por {} guaranies"  
print(orden.format(cantidad, item, precio))
```

Cadenas o Texto(str)

https://www.w3schools.com/python/python_strings_methods.asp

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string
format_map()	Formats specified values in a string
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isascii()</u>	Returns True if all characters in the string are ascii characters
<u>isdecimal()</u>	Returns True if all characters in the string are decimals
<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier

Ejemplo 1

Calcular la cantidad de alumnos que obtuvieron nota inferior al promedio del curso en cierta materia. Hay N alumnos (donde N es un valor entero positivo ingresado por teclado), y todos rindieron. Las notas son números enteros que van del 0 al 100 (se asume que todas las notas son correctas).

¿Cuáles son los pasos para resolverlo?

Ejemplo 1

#1- Se ingresa el tamaño del vector

```
N = int(input("Ingrese la cantidad de alumnos: "))
```

#2- Se define y carga la lista

```
notas=[]  
for i in range(N):  
    nota = int(input(f"Ingrese la nota {i}: "))  
    notas.append(nota)
```

#3- Se calcula el promedio

```
suma = 0 #acumulador  
for nota in notas:  
    suma+=nota  
promedio = suma/N  
print(f"El promedio es: {promedio:.2f}")
```

#4- Se determinan las notas menores que el promedio

```
menorProm = []  
for nota in notas:  
    if nota<promedio:  
        menorProm.append(nota)  
print(f"Hay {len(menorProm)} alumnos con nota inferior al promedio:")  
print(menorProm)
```

Ejemplo 2

Se tienen anotadas las temperaturas (promedio) de todos los días del mes de febrero de 2022 y se deben almacenar en el vector `tempFeb`. Diseñar un programa que obtenga las temperaturas máxima, mínima (e indique los días correspondientes), y el promedio de las que se encuentran entre los días 21 y 27.

Ejemplo 2

#1- Se define y carga la lista

```
N = 28
tempFeb=[]
for i in range(N):
    t = int(input(f"Ingrese la temperatura del dia {i+1}: "))
    tempFeb.append(t)
```

#2- Se determinan los máximos y mínimos

```
posMax=0 #Posición de la temperatura máxima
posMin=0 #Posición de la temperatura mínima
for i in range(1,N):
    if tempFeb[i]>tempFeb[posMax]:
        posMax=i
    if tempFeb[i]<tempFeb[posMin]:
        posMin=i
print(f"El dia más frío fue el {posMin+1}/Feb con {tempFeb[posMin]} grados")
print(f"El dia más caluroso fue el {posMax+1}/Feb con {tempFeb[posMax]} grados")
```

#3- Se calcula el promedio

```
suma=0
for i in range(20,27):
    suma+=tempFeb[i]
prom=suma/7
print(f"El promedio de los días 21 al 27 fue de {prom:.2f} grados")
```


Listas y funciones

Se puede pasar una lista como argumento de una función. Un ejemplo es el siguiente:

```
def mostrarLista(lista):  
    print("Los elementos de la lista son:")  
    for x in lista:  
        print(x,end=" ")
```

```
A = [1,2,3,4,5,6]  
mostrarLista(A)
```

```
Los elementos de la lista son:  
1 2 3 4 5 6
```

Ejemplos 3 y 4

- Diseñar un programa que calcule la magnitud de un vector de N elementos (números reales) que están en una lista A . Este programa debe contar con una función **magnitud** que reciba la lista y devuelva la magnitud del vector.
- Diseñar un programa que calcule el producto escalar de dos vectores A y B de N elementos (números reales), estando cada uno de ellos representados en una lista. Este programa debe contar con una función **escalar** que reciba estas dos y devuelva el producto escalar.

Recordar: Paso por valor

```
a = int(input("Ingrese a: "))  
b = int(input("Ingrese b: "))  
c = suma(a,b)  
print(c)
```

Función suma

```
def suma(x,y):  
    c = x+y  
    return c
```

Memoria

10 15
a b

suma 10 15
x y

Listas y funciones

Cuando una lista es el argumento de una función, se pasa la **referencia** de la lista al llamar a dicha función (es decir, se pasa la lista propiamente dicha). Por ello, si se hacen cambios dentro de la función, éstos se reflejarán en la lista original.

```
def mostrarLista(lista):  
    print("Los elementos de la lista son:")  
    for x in lista:  
        print(x,end=" ")
```

```
def sumar10(lista):  
    for i in range(len(lista)):  
        lista[i]+=10
```

```
A = [1,2,3,4,5,6]  
sumar10(A)  
mostrarLista(A)
```

```
Los elementos de la lista son:  
11 12 13 14 15 16
```

Ejemplo 5

Escribir una función que reciba una lista y que pase cada elemento a la posición derecha. El elemento final de la lista pasa al inicio.

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



9	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

Tuplas en Python

Una **tupla** consiste de un número de valores separados por comas. Las tuplas son *inmutables* (sus elementos no pueden modificarse*). Comparten muchas propiedades con las listas, y pueden tener elementos heterogéneos.

```
>>> t=1,2,3
```

```
>>> t
```

```
(1, 2, 3)
```

```
>>> t=(1,2,3)
```

```
>>> t
```

```
(1, 2, 3)
```

```
>>> t[1]
```

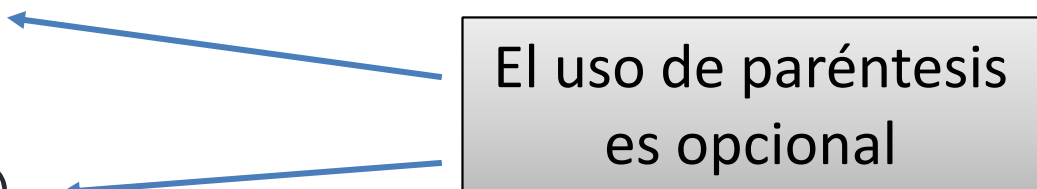
```
2
```

```
>>> t[1]=5
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item  
assignment
```



El uso de paréntesis
es opcional

Retorno de múltiples valores (tupla)

Una función puede retornar una tupla. De esta forma, una función puede retornar múltiples valores.

```
def sumaYMult(a,b):  
    suma = a+b  
    multiplicacion = a*b  
    return suma, multiplicacion #se retorna una tupla
```

```
a=5
```

```
b=6
```

```
#El primer valor de la tupla se guarda en s
```

```
#Mientras que el segundo, en m
```

```
s, m = sumaYMult(a,b)
```

```
print("La suma es:",s)
```

```
print("La multiplicacion es:",m)
```

Conjuntos (*sets*)

Un **conjunto** es una colección *desordenada* que no posee elementos duplicados. Los objetos *set* soportan operaciones matemáticas como unión, intersección y diferencia.

```
>>> materias = {"Física", "Estadística", "Álgebra Lineal",  
"Cálculo", "Física", "Programación"}
```

```
>>> materias  
{'Álgebra Lineal', 'Programación', 'Física', 'Cálculo',  
'Estadística'}
```

```
>>> materias[0]
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#4>", line 1, in <module>
```

```
    materias[0]
```

```
TypeError: 'set' object is not subscriptable
```


Operaciones con conjuntos

```
>>> materias2 = {"Química", "Estadística", "Geometría",  
"Física"}
```

Unión

```
materias | materias2  
{'Álgebra Lineal', 'Geometría',  
'Programación', 'Química', 'Física',  
'Cálculo', 'Estadística'}
```

Intersección

```
materias & materias2  
{'Física', 'Estadística'}
```

Diferencia

```
materias - materias2  
{'Álgebra Lineal', 'Cálculo',  
'Programación'}
```

Algunas operaciones con conjuntos

<code>s.add(x)</code>	Agrega el elemento <code>x</code> al conjunto <code>s</code>
<code>s.remove(x)</code>	remueve <code>x</code> del conjunto <code>s</code> ; lanza un error <code>KeyError</code> si <code>x</code> no existe
<code>s.discard(x)</code>	remueve <code>x</code> del conjunto <code>s</code> si esta presente
<code>s.pop()</code>	remueve y retorna un element arbitrario de <code>s</code> ; lanza un error <code>KeyError</code> si <code>s</code> esta vacio
<code>s.clear()</code>	remueve todos los elementos de <code>s</code>

* Traducido de <https://docs.python.org/2/library/sets.html>

```
materias2 = {"Química", "Estadística", "Geometría", "Física"}
```

Diccionarios

Un **diccionario** es una lista desordenada de *claves* y *valores* asociados, con el requerimiento de que las claves sean únicas (dentro de un diccionario).

'Colbes'	:	'0971445566'
'Insfrán'	:	'0981112233'
'Parra'	:	'0983998877'
'Riveros'	:	'0991556632'

```
agenda =  
{ 'Colbes': '0971445566',  
  'Insfran': '0981112233',  
  'Parra': '0983998877',  
  'Parra': '961998877' }
```



Separa una clave y el valor asociado

Claves: inmutables **Valores:** cualquier tipo

Acceder a valores en un diccionario

Para acceder a un valor del diccionario, en lugar de colocar una posición (como en el caso de las listas) se coloca su clave.

```
>>>print('El número de teléfono de José es:', agenda['Colbes'])  
El número de teléfono de José es: 0971445566
```

Si se intenta acceder a una clave que no está en el diccionario, dará un error:

```
>>>print('El número de teléfono de María es:', agenda['Pérez'])  
Traceback (most recent call last):  
  File "<pyshell#6>", line 1, in <module>  
    print('El número de teléfono de María es:', agenda['Pérez'])  
KeyError: 'Pérez'
```

Para ello, se puede verificar primero:

```
>>>'Pérez' in agenda  
False
```

Actualizar un diccionario

Agregar un par *clave:valor* al diccionario:

```
>>>agenda['Pérez'] = '0972999555'  
>>>agenda  
{'Colbes': '0971445566', 'Insfrán': '0981112233', 'Parra':  
'0983998877', 'Riveros': '0991556632', 'Pérez': '0972999555'}
```

Modificar un elemento existente:

```
>>>agenda['Colbes'] = '0983948576'  
>>>agenda  
{'Colbes': '0983948576', 'Insfrán': '0981112233', 'Parra':  
'0983998877', 'Riveros': '0991556632', 'Pérez': '0972999555'}
```

Actualizar un diccionario

Borrar un elemento del diccionario:

```
>>>del agenda['Insfrán']  
>>>agenda  
{'Colbes': '0983948576', 'Parra': '0983998877', 'Riveros':  
'0991556632', 'Pérez': '0972999555'}
```

Añadir un diccionario a otro:

```
>>>agenda_vieja = {'Mamá': '0981567890', 'Papá': '0974521478'}  
>>>agenda.update(agenda_vieja)  
>>>agenda  
{'Colbes': '0983948576', 'Parra': '0983998877', 'Riveros':  
'0991556632', 'Pérez': '0972999555', 'Mamá': '0981567890',  
'Papá': '0974521478'}
```

Listar elementos del diccionario

El tamaño del diccionario puede ser obtenido con la función **len()** (como en las listas):

```
>>>len(agenda)
6
```

Es posible obtener la lista de las claves de un diccionario (o de los valores), con la ayuda de la función **list()**:

```
>>>list(agenda.keys())
['Colbes', 'Parra', 'Riveros', 'Pérez', 'Mamá', 'Papá']
>>>list(agenda.values())
['0983948576', '0983998877', '0991556632', '0972999555',
'0981567890', '0974521478']
```

Ejercicio propuesto

Se tiene una lista con números binarios de tamaño n (siendo el mismo un múltiplo de 3). Un ejemplo es el siguiente:

1	0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

Diseñar una función que cree una nueva lista a partir de la lista que recibe, donde después de cada 3 elementos de la lista original, se agregue un elemento que indique la cantidad de 1's en esos tres elementos. La función debe retornar esta lista creada. En nuestro caso, la lista generada sería:

1	0	1	2	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---

Solución

```
def conteode1(a):  
    c = 0  
    b = []  
    x = 1  
    for i in a:  
        b.append(i)  
        if i==1:  
            c += 1  
        if x%3==0:  
            b.append(c)  
            c=0  
        x += 1  
    return b
```

```
def conteo_unos(a):  
    nuevo = []  
    for i in range(0, len(a), 3):  
        # Toma 3 elementos  
        sub = a[i:i + 3]  
        #contamos 1  
        unos = sub.count(1)  
  
        nuevo.extend(sub)  
        nuevo.append(unos)  
    return nuevo
```

Gracias por la atención

