

DESCUBRIENDO EL PODER DE LA PROGRAMACION

CURSO INICIAL DE PYTHON

Jimmy Chung
Alexander Solis



CONTENIDO DEL CURSO

1. **Introducción, Instalación y Conceptos básicos**
2. **Variables, Expresiones, Funciones y Operadores**
3. **Condicionales y Ciclos – Patrones de sintaxis válidos**
4. **Listas, tuplas y diccionarios**
5. **Errores y Excepciones**
6. **Clases y funciones en Python**



Tema 3: Listas, Tuplas y Diccionarios



Funciones «built-in»

Funciones «built-in»: Incorporadas por defecto en el propio lenguaje

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	any()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<https://docs.python.org/es/3/library/functions.html?highlight=built>



LISTAS (list)

En Python, una lista es una estructura de datos que permite almacenar y organizar múltiples elementos en una secuencia ordenada. Las listas son muy versátiles y se utilizan con frecuencia en programación debido a su capacidad para almacenar diferentes tipos de datos y su flexibilidad para realizar operaciones y manipulaciones.

```
lista_1 = [1, 2, 3, 4] # Hemos creado una lista de números.  
lista_2 = list("1234") # Se puede crear una lista usando list() y pasando un objeto iterable.  
lista_3 = [1, "Hola", 3.67, [1, 2, 3]] # Pueden almacenar tipos de datos distintos  
print(lista_3[0]) # Imprime 1  
print(lista_3[1]) # Imprime Hola  
print(lista_3[-1]) # Imprime [1, 2, 3]
```



LISTAS: Características

- Secuencia ordenada: Los elementos en una lista se almacenan en un orden específico y se pueden acceder utilizando su posición, también conocida como índice. Ejemplo: `segundo_elemento = lista[1]`
- Mutable: Las listas son objetos mutables, lo que significa que se pueden modificar después de su creación. Es posible agregar, eliminar o modificar elementos en una lista. Ejemplo: `lista[3] = "Modificado"`
- Almacenamiento heterogéneo: Las listas pueden contener elementos de diferentes tipos de datos, como enteros, cadenas, flotantes, booleanos, incluso otras listas u objetos personalizados. Ejemplo: `lista = [1, "Hola", 3.67, [1, 2, 3]]`
- Tamaño variable: Las listas pueden cambiar de tamaño dinámicamente a medida que se agregan o eliminan elementos. No es necesario especificar un tamaño fijo al crear una lista



LISTAS: Operaciones comunes

- Agregar elementos: Podemos agregar elementos a una lista utilizando el método `append()` para agregar un elemento al final, o el operador `+` para concatenar dos listas.
- Eliminar elementos: Puedes eliminar elementos de una lista utilizando los métodos `remove()` para eliminar un elemento específico, o `pop()` para eliminar un elemento en una posición dada.
- Obtener la longitud: Puedes obtener la longitud de una lista utilizando la función `len()`.
- Recorrer una lista: Puedes recorrer una lista utilizando bucles como `for` o `while`, y acceder a cada elemento mediante su índice.



LISTAS: Principales métodos

- **append():** Añade un elemento al final de la lista.
Sintaxis: `append(<obj>)`

```
l = [1, 2]
l.append(3)
print(l) #[1, 2, 3]
```
- **extend():** Añade una lista a la lista inicial
Sintaxis: `extend(<iterable>)`

```
l = [1, 2]
l.extend([3, 4])
print(l) #[1, 2, 3, 4]
```
- **insert():** Añade un elemento en una posición o índice determinado
Sintaxis: `insert(<index>, <obj>)`

```
l = [1, 3]
l.insert(1, 2)
print(l) #[1, 2, 3]
```
- **remove():** Recibe como argumento un objeto y lo borra de la lista.
Sintaxis: `remove(<obj>)`

```
l = [1, 2, 3]
l.remove(3)
print(l) #[1, 2]
```
- **count():** Cuenta el número de veces que el objeto pasado como parámetro se ha encontrado en la lista.



LISTAS: Principales métodos

- `pop()`: Elimina un elemento de la lista dando su índice, si no se especifica el índice, borra el último elemento de la lista.

Sintaxis: `pop(index=-1)`.

```
lista = [1, 2, 3]
lista.pop() #Elimina el ultimo elemento de la lista, ya que no se especifica el índice.
print(lista) #[1, 2]
```

```
lista = [1, 2, 3]
lista.pop(1) #Elimina el segundo elemento de la lista, el primero es el cero(0).
print(lista) #[1, 3]
```

- `reverse()`: Invierte el orden de la lista.

Sintaxis: `reverse()`

```
l = [1, 2, 3]
l.reverse()
print(l) #[3, 2, 1]
```



LISTAS: Principales métodos

- `sort()`: Ordena los elementos de menos a mayor por defecto.

```
lista = [3, 1, 2]
lista.sort()
print(lista) #[1, 2, 3]
```

Y también permite ordenar de mayor a menor si se pasa como parámetro `reverse=True`.

```
lista = [3, 1, 2]
lista.sort(reverse=True)
print(lista) #[3, 2, 1]
```

- `clear()`: Elimina todos los elementos de la lista. Equivalente a `del lista[:]`
- `index()`: Recibe como parámetro un objeto y devuelve el índice o posición en la que se encuentra el objeto en la lista.

Sintaxis: `index(<obj>, index)` #El parámetro `index` es opcional, e indica a partir de qué índice empezamos a buscar.

```
lista = ["Periphery", "Intervals", "Monuments"]
print(lista.index("Intervals"))#1
```

```
lista = [1, 1, 1, 1, 2, 1, 4, 5]
print(lista.index(1, 4)) #5
```



TUPLAS (tuple)

Las tuplas son una estructura de datos en Python muy similares a las listas, pero a diferencia de las listas, las tuplas son inmutables, lo que significa que no se pueden modificar una vez creadas. Se representan utilizando paréntesis ().

Las tuplas son útiles cuando deseamos almacenar un conjunto de valores que no cambiarán a lo largo del tiempo, como las coordenadas de un punto en un plano, los días de la semana o los meses del año.

```
tupla = (1, 2, 3, 4, 5) # Hemos creado una tupla.
```

```
tupla_2 = tuple("12345") # Hemos creado una tupla usando tuple().
```

```
tupla_3 = (1, "Hola", 3.67, [1, 2, 3]) # Pueden almacenar tipos de datos distintos
```



TUPLAS: Operaciones comunes

Con las tuplas podemos realizar las mismas operaciones que hacemos con una lista, excepto aquellas que la modifican: `reverse()`, `append()`, `extend()`, `remove()`, `clear()`, `sort()`.

Pero si queremos ordenar o invertir una tupla, podemos generar una nueva tupla a partir de una existente, utilizando los métodos `sorted()` o `reversed()`.

```
tupla = (3, 1, 2)
tupla_ordenada = sorted(tupla)
print(tupla) #[3, 1, 2]
print(tupla_ordenada) #[1, 2, 3]
```

```
tupla = (1, 2, 3, 4)
tupla_reversada = tuple(reversed(tupla))
print(tupla_reversada) #[4, 3, 2, 1]
```

También podemos asignar el valor de una tupla con n elementos a n variables:

```
tupla = ('Melchor', 'Gaspar', 'Baltasar')
rey_1, rey_2, rey_3 = tupla #A esta operación también se le conoce como desempaquetado de tuplas
print(rey_1) #Melchor
print(rey_2) #Gaspar
print(rey_3) #Baltasar
```



DICCIONARIOS (dict)

Los diccionarios son una estructura de datos en Python que permite almacenar pares clave-valor. A diferencia de las listas y las tuplas, que son secuencias ordenadas, los diccionarios son estructuras no ordenadas y se accede a sus elementos a través de claves en lugar de índices. Los diccionarios se representan utilizando llaves {} y los pares clave-valor se separan por dos

```
puntos = {  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
}  
print(diccionario) #{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```



DICCIONARIOS (dict)

Otra forma de crear un diccionario en Python es usando la función built-in `dict()` e introduciendo los pares key: value entre paréntesis.

```
diccionario = dict({  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
})  
print(diccionario) #{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```

También es posible usar `dict()` como constructor para crear un diccionario:

```
diccionario = dict(Nombre="Sara", Edad=27, Documento=1003882)  
print(diccionario) #{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```



DICCIONARIOS (dict)

Los diccionarios pueden ser anidados, es decir que el valor de un item de un diccionario, puede ser otro diccionario.

```
diccionario = {  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882,  
    "Direccion": {  
        "Direccion": "Calle 100 # 100-24",  
        "Ciudad": "Bogota",  
        "Pais": "Colombia"  
    }  
}
```



DICCIONARIOS: Operaciones

Acceder a los elementos de un diccionario:

Se puede acceder a sus elementos con [] o también con la función `get()`

```
diccionario = dict(  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
)
```

```
print(diccionario['Nombre']) #Sara  
print(diccionario.get('Nombre')) #Sara
```



DICCIONARIOS: Operaciones

Modificar los elementos de un diccionario:

Para modificar un elemento basta con usar [] con el nombre del key y asignar el valor que queremos:

```
diccionario = dict(  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
)
```

```
diccionario['Nombre'] = "Laura"  
diccionario['Direccion'] = "Calle 100 # 23-24" #Si el key no existe, se añade automáticamente  
print(diccionario)  
#{'Nombre': 'Laura', 'Edad': 27, 'Documento': 1003882, 'Direccion': 'Calle 100 # 23-24'}
```



DICCIONARIOS: Operaciones

Iterar un diccionario:

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos:

```
diccionario = dict({  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
})
```

Imprime los key del diccionario

```
for llave in diccionario:
```

```
    print(llave)
```

#Nombre

#Edad

#Documento

Imprime los value del diccionario

```
for llave in diccionario:
```

```
    print(diccionario[llave])
```

#Sara

#27

#1003882



DICCIONARIOS: Operaciones

Iterar un diccionario:

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos:

```
diccionario = dict({  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
})  
# Imprime los key, value del diccionario  
for llave, valor in diccionario.items():  
    print(llave)  
#Nombre Laura  
#Edad 27  
#Documento 1003882
```



DICCIONARIOS: Operaciones

Iterar un diccionario:

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos:

```
diccionario = dict({  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
})  
# Imprime los key, value del diccionario  
for llave, valor in diccionario.items():  
    print(llave, valor)  
#Nombre Laura  
#Edad 27  
#Documento 1003882
```



DICCIONARIOS: Métodos

`clear()`: El método `clear()` elimina todo el contenido del diccionario.

```
diccionario = { "Nombre": "Sara", "Edad": 27 }  
diccionario.clear()  
print(diccionario) #{}
```

`get(<key>[, <default>])`: Permite consultar el `value` para un `key` determinado. El segundo parámetro es opcional, y en el caso de proporcionarlo es el valor a devolver si no se encuentra la `key`.

```
diccionario = { "Nombre": "Sara", "Edad": 27 }  
print(diccionario.get('Edad')) #27  
print(diccionario.get('Profesion', 'Sin profesion')) #Sin profesion
```



DICCIONARIOS: Métodos

items(): Devuelve una lista con los **keys** y **values** del diccionario. Si puede convertir en una lista de tuplas donde los primeros elementos son las **key** y los segundos los **value**.

```
diccionario = {'a': 1, 'b': 2}
it = diccionario.items()
print(it) #dict_items([('a', 1), ('b', 2)])
print(list(it)) #[('a', 1), ('b', 2)]
print(list(it)[0][0]) #a
```

keys(): Devuelve una lista con todas las keys del diccionario.

```
diccionario = {'a': 1, 'b': 2}
llaves = diccionario.keys()
print(llaves) #dict_keys(['a', 'b'])
print(list(llaves)) #['a', 'b']
```



DICCIONARIOS: Métodos

values(): Devuelve una lista con los **values** del diccionario.

```
diccionario = {'a': 1, 'b': 2}
print(list(diccionario.values())) #[1, 2]
```

pop(<key>[,<default>]): Busca y elimina la **key** que se pasa como parámetro y devuelve su valor asociado. Daría un error si se intenta eliminar una **key** que no existe. Pero si pasamos el segundo parámetro (**default**) que es el valor a devolver si la **key** no se ha encontrado, no se genera error.

```
diccionario = {'a': 1, 'b': 2}
valor = diccionario.pop('a')
print(diccionario) #{'b': 2}
print(valor) #1
```

```
diccionario = {'a': 1, 'b': 2}
valor = diccionario.pop('c', -1)
print(diccionario) #{'a':1, 'b': 2}
print(valor) #-1
```



DICCIONARIOS: Métodos

popitem(): Elimina y retorna una pareja (**key**, **value**) del diccionario. Elimina en orden LIFO (Last In - First Out): Primero en entrar - Último en salir

```
diccionario = {'a': 1, 'b': 2}
pareja = diccionario.popitem()
print(diccionario) #{'a': 1}
print(pareja) #('b': 2)
```

update(<obj>): El método **update()** se llama sobre un diccionario y tiene como entrada otro diccionario. Los **value** son actualizados y si alguna **key** del nuevo diccionario no esta, es añadida.

```
d1 = {'a': 1, 'b': 2}
d2 = {'a': 0, 'd': 400}
d1.update(d2) #Asigna los valores d2 a d1
print(d1) #{'a': 0, 'b': 2, 'd': 400}
```



SET o Conjunto

Los set en Python son un tipo que permite almacenar varios elementos y acceder a ellos de una forma muy similar a las [listas](#) pero con ciertas diferencias:

- Los elementos de un set son único, lo que significa que no puede haber elementos duplicados.
- Los set son desordenados, lo que significa que no mantienen el orden de cuando son declarados.
- Sus elementos deben ser inmutables (No se pueden modificar).
- Son muy útiles para almacenar una colección de elementos únicos y realizar operaciones matemáticas de conjuntos, como intersección, unión, diferencia.

```
s = set([5, 4, 6, 8, 8, 1])  
print(s) #{1, 4, 5, 6, 8}  
print(type(s)) #<class 'set'>
```

```
s = {5, 4, 6, 8, 8, 1}  
print(s) #{1, 4, 5, 6, 8}  
print(type(s)) #<class 'set'>
```



SET o Conjunto: Métodos

`add(<elem>)`: Permite añadir un elemento al set

```
l = set([1, 2])  
l.add(3)  
print(l) #{1, 2, 3}
```

`remove(<elem>)`: Elimina el elemento que se pasa como parámetro. Si no se encuentra, se lanza la excepción `KeyError`

```
s = set([1, 2])  
s.remove(2)  
print(s) #{1}
```

`discard(<elem>)`: Elimina el elemento que se pasa como parámetro, y si no se encuentra no hace nada.

```
s = set([1, 2])  
s.discard(3)  
print(s) #{1, 2}
```


`pop()`: Elimina un elemento aleatorio del set (igual que en los diccionarios).

`clear()`: Elimina todos los elementos de set (igual que en en los diccionarios).



Anexo: Comparativo list, set, tuple, dict

Python 3 · Algunas operaciones con estructuras de datos

	LISTA	TUPLA	CONJUNTO	DICCIONARIO
Características	Datos heterogéneos. Acepta repetidos. Elementos mutables y accesibles por índice.	Datos heterogéneos. Acepta repetidos. Elementos inmutables y accesibles por índice.	Datos heterogéneos. Elementos de tipos inmutables, no accesibles por índice. No admite repetidos.	Claves de tipos inmutables, únicas (no admite repetidas). Valores de cualquier tipo, accesibles por clave, admiten repetidos. Pares heterogéneos.
Crear	<code>[]</code> ó <code>list()</code>	<code>()</code> ó <code>tuple()</code>	<code>set()</code>	<code>{}</code> ó <code>dict()</code>
Inicializar con datos	<ul style="list-style-type: none"> <code>[elemento1, elemento2, elemento3]</code> <code>list(iterable)</code> para crear con los elementos de una secuencia o contenedor. 	<ul style="list-style-type: none"> <code>elemento1, elemento2, elemento3</code> (los paréntesis son opcionales) <code>tuple(iterable)</code> para crear con los elementos de una secuencia o contenedor. 	<ul style="list-style-type: none"> <code>{elemento1, elemento2, elemento3}</code> <code>set(iterable)</code> para crear con los elementos de una secuencia o contenedor. 	<ul style="list-style-type: none"> <code>{clave1: valor1, clave2: valor2, clave3: valor3}</code> <code>dict([(c1,v1), (c2,v2)])</code>
Insertar elementos	<ul style="list-style-type: none"> <code>lista.append(elemento)</code> agrega elemento al final <code>lista.insert(posición, elemento)</code> inserta elemento en posición del índice. 	Sólo es posible inicializarla en la creación. Luego no se puede agregar, porque es inmutable.	<code>conjunto.add(elemento)</code> donde elemento es un solo dato.	<ul style="list-style-type: none"> <code>dicc[clave]=valor</code> <code>dicc.update({c3:v3, c4:v4})</code>
Acceder a un elemento	<ul style="list-style-type: none"> <code>lista[indice]</code> lectura/escritura. <code>lista[inicio:fin:paso]</code> Iterando por sus elementos. 	<ul style="list-style-type: none"> <code>tupla[indice]</code> sólo lectura <code>tupla[inicio:fin:paso]</code> Iterando por sus elementos. 	Iterando por sus elementos (no soporta índices). No es posible modificar elementos.	<ul style="list-style-type: none"> <code>dicc[clave]</code> <code>dicc.get(clave, val_defecto)</code> Iterando por sus elementos.
Iterar usando el índice	<code>for i in range(len(lista)):</code> <code>print(lista[i])</code>	<code>for i in range(len(tupla)):</code> <code>print(tupla[i])</code>	No tiene índice.	No tiene índice (el "índice" son las claves).
Iterar con for para iterables	<code>for elemento in lista</code>	<code>for elemento in tupla</code>	<code>for elemento in conjunto</code>	<ul style="list-style-type: none"> <code>for clave in dicc.keys()</code> <code>for valor in dicc.values()</code> <code>for c,v in dicc.items()</code>
Pertenencia	<code>elemento in lista</code>	<code>elemento in tupla</code>	<code>elemento in conjunto</code>	<ul style="list-style-type: none"> <code>clave in dicc.keys()</code> <code>value in dicc.values()</code>
Eliminar elemento	<ul style="list-style-type: none"> <code>del lista[indice]</code> <code>lista.remove(elemento)</code> elimina la primera ocurrencia 	No es posible porque son inmutables.	<ul style="list-style-type: none"> <code>conjunto.remove(elemento)</code> <code>conjunto.discard(elemento)</code> 	<code>del dicc[clave]</code>
Cantidad de elementos	<code>len(lista)</code>	<code>len(tupla)</code>	<code>len(conjunto)</code>	<code>len(dicc)</code>
Vaciar	<code>lista.clear()</code>	No es posible porque son inmutables.	<code>conjunto.clear()</code>	<code>dicc.clear()</code>



List comprehensions

Las List comprehensions (comprensiones de listas) en Python son una forma concisa y poderosa de crear listas de manera elegante y eficiente. Proporcionan una sintaxis compacta para generar listas basadas en una expresión y una o varias iteraciones.

Sintaxis básica de una List comprehension:

```
nueva_lista = [expresion for elemento in secuencia]
```

Donde:

nueva_lista: Es la lista resultante generada por la List comprehension.

expresion: Es una expresión que define cómo se calculará cada elemento de la lista resultante.

elemento: Es una variable que representa cada elemento de la secuencia en la que se está iterando.

secuencia: Es una secuencia, como una lista, tupla o rango, sobre la cual se realizará la iteración.



List comprehensions

```
cuadrados = [i**2 for i in range(5)] #[0, 1, 4, 9, 16]
```

Si no existieran las List comprehensions, podríamos hacer lo mismo de la siguiente forma, pero necesitamos más líneas de código.

```
cuadrados = []  
for i in range(5):  
    cuadrados.append(i**2)  
#[0, 1, 4, 9, 16]
```

La expresión también puede ser una llamada a una función:

```
def eleva_al_2(i):  
    return i**2  
  
cuadrados = [eleva_al_2(i) for i in range(5)]  
#[0, 1, 4, 9, 16]
```



List comprehensions

Las List comprehensions también pueden incluir cláusulas opcionales como `if` para filtrar elementos o realizar condiciones adicionales.

```
nueva_lista = [expresion for elemento in secuencia if condicion]
```

En este caso, la `condicion` se evalúa para cada `elemento` de la `secuencia` y solo se incluyen en la lista resultante aquellos elementos que cumplen con la condición especificada.

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
cuadrados_pares = [num ** 2 for num in numeros if num % 2 == 0]

print(cuadrados_pares)
# Salida: [4, 16, 36, 64, 100]
```

List comprehensions para crear una lista de los cuadrados de los números pares de 1 a 10



Ejercicios

- Escribe un programa que reciba una lista de números y devuelva la suma de todos los elementos.
- Dada una lista de palabras, crea una nueva lista que contenga únicamente las palabras que tengan más de 5 caracteres.
- Escribe una función que tome una lista de números y devuelva una nueva lista con los elementos ordenados de forma ascendente.
- Dada una lista de números, encuentra el número más grande y el número más pequeño.



Ejercicios

- Dada una lista de nombres, elimina todos los nombres duplicados y devuelve la lista sin duplicados.
- Escribe un programa que tome una lista de números y devuelva una lista con los números pares.
- Dada una lista de números, calcula la media (promedio) de todos los elementos.
- Escribe una función que tome una lista de cadenas y devuelva una nueva lista con las cadenas en orden inverso.
- Dada una lista de números, elimina todos los números impares y devuelve la lista modificada.



Ejercicios

- Escribe un programa que tome dos listas y devuelva una nueva lista que contenga los elementos comunes entre ambas listas.
- Escribe un programa que tome una tupla de números y devuelva la suma de todos los elementos.
- Dada una lista de tuplas, cada una conteniendo un nombre y una edad, ordena la lista de tuplas por edad de forma ascendente.
- Escribe una función que tome una tupla de números y devuelva una nueva tupla con los números ordenados de forma descendente.



Ejercicios

- Escribe un programa que tome dos tuplas de números y devuelva una nueva tupla que contenga la suma de los elementos correspondientes en cada posición.
- Escribe un programa que tome un diccionario de nombres y edades y devuelva el nombre de la persona más joven.
- Dada una lista de diccionarios que contienen información de empleados (nombre, salario), calcula el salario promedio de todos los empleados.



Ejercicios

- Escribe una función que tome un diccionario de palabras y sus traducciones, y permita al usuario buscar la traducción de una palabra específica.
- Dado un diccionario de productos y sus precios, crea una función que encuentre el producto más caro y devuelva su nombre y precio.
- Escribe un programa que tome dos diccionarios y los combine en uno solo, manteniendo las claves y sumando los valores si hay claves repetidas.



Ejercicios

- Dado un diccionario de nombres y sus calificaciones, crea una función que devuelva el nombre del estudiante con la calificación más alta.
- Escribe un programa que tome un diccionario de palabras y cuente cuántas veces aparece cada palabra.



Nuestro tercer proyecto

Registro de estudiantes.

El programa debe permitir al usuario realizar las siguientes acciones:

1. Agregar registro de estudiante.
2. Mostrar registro de estudiante.
3. Mostrar registros de estudiantes.
4. Eliminar registro de estudiante.
5. Cambiar de curso a un estudiante.



Referencias bibliográficas

Documentación oficial Python.

<https://docs.python.org/es/3/tutorial/index.html>

<https://docs.python.org/es/3/tutorial/datastructures.html#more-on-lists>

<https://docs.python.org/es/3/tutorial/datastructures.html#tuples-and-sequences>

<https://docs.python.org/es/3/tutorial/datastructures.html#dictionaries>

<https://docs.python.org/es/3/tutorial/datastructures.html#sets>

