COMANDOS
listas=[]
- mutables

count = cuantas veces hay ese objeto

```
lista_nueva = [1, 2, 3, 4, 5]

lista_nueva.append(3)

print(lista_nueva)

print(lista_nueva.count(3))
```

```
[1, 2, 3, 4, 5, 3]
2
```

```
print(lista_nueva.index(4))
```

```
[1, 2, 3, 4, 5, 3]
2
3
```

index devuelve la posicion del 1er elemento

```
lista_nueva.remove(3)

print(lista_nueva)
```

```
[1, 2, 3, 4, 5, 3]
2
3
[1, 2, 4, 5, 3]
```

remove quita la 1ra posición de ese elemento
TUPLA=()
- ordenada
- heterogenea

CONJUNTOS={}
- no ordenados
- mutables
- noo se repite

```
# Conjunto

print(set([5, 2, 5, 1, 1.5]))
print(set((5, 2, 5, 1, 1.5)))
print(set("52511.5"))
```

```
{1, 2, 5, 1.5}
{1, 2, 5, 1.5}
{'.', '5', '1', '2'}
```

```
conjunto_2 = set([5, 3, 5, 6])
conjunto_3 = set([4, 2])
print(conjunto, conjunto_2, conjunto_3)
print(conjunto.intersection(conjunto_2))
```

```
{2, 3, 4} {3, 5, 6} {2, 4}
{3}
```

intersection es el punto donde las 2 coinciden
o utilizar &

```
print(conjunto & conjunto_2)
```

```
print(conjunto_2.issubset(conjunto))
print(conjunto_3.issubset(conjunto))
```

issubset si los elementos de un conjuntop estan en otro conjunto
&
|
&

DICCIONARIO= {clave:valor}

```python
diccionario = {1: "Uno", 2: "Dos"}
diccionario[3] = "Tres"
print(diccionario)

dict_lista_tuplas = dict([(1, "Uno"), (2, "Dos"), (3, "Tres")])
print(dict_lista_tuplas)

dict_lista_string = dict(Uno = 1, Dos = 2, Tres = 3)
print(dict_lista_string)
```

```
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{'Uno': 1, 'Dos': 2, 'Tres': 3}
```

```python
diccionario = {1: "Uno", 2: "Dos"}
diccionario[3] = "Tres"
print(diccionario)

dict_lista_tuplas = dict([(1, "Uno"), (2, "Dos"), (3, "Tres")])
print(dict_lista_tuplas)

dict_lista_string = dict(Uno = 1, Dos = 2, Tres = 3)
print(dict_lista_string)

dict_tipos = {1: "integer", 2.2: "float", "texto": "string", (1, 2): "tupla"}
print(dict_tipos)

dict_repeticion = {1: "Primero", 1: "Último"}
print(dict_repeticion)

print(diccionario, diccionario.keys(), diccionario.values(), diccionario.items())
claves = diccionario.values()
print(claves)
diccionario[1] = "One"
print(claves)
```

```
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{'Uno': 1, 'Dos': 2, 'Tres': 3}
{1: 'integer', 2.2: 'float', 'texto': 'string', (1, 2): 'tupla'}
{1: 'Último'}
{1: 'Uno', 2: 'Dos', 3: 'Tres'} dict_keys([1, 2, 3]) dict_values(['Uno', 'Dos', 'Tres']) dict_items([(1, 'Uno'), (2, 'Dos'), (3, 'Tres')])
dict_values(['Uno', 'Dos', 'Tres'])
dict_values(['One', 'Dos', 'Tres'])
```

```python
print(claves)
diccionario.pop(2)
print(diccionario)
```

```
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{1: 'Uno', 2: 'Dos', 3: 'Tres'}
{'Uno': 1, 'Dos': 2, 'Tres': 3}
{1: 'integer', 2.2: 'float', 'texto': 'string', (1, 2): 'tupla'}
{1: 'Último'}
{1: 'Uno', 2: 'Dos', 3: 'Tres'} dict_keys([1, 2, 3]) dict_values(['Uno', 'Dos', 'Tres']) dict_items([(1, 'Uno'), (2, 'Dos'), (3, 'Tres')])
dict_values(['Uno', 'Dos', 'Tres'])
dict_values(['One', 'Dos', 'Tres'])
{1: 'One', 3: 'Tres'}
```

len saber el total de posiciones

```
s = 'nombre,edad,carrera,matricula'
partes = s.split(',')
```

.split considera los espacios

startswith

Regresa verdadero si una cadena empieza con una subcadena
print('hola mundo'.startswith('hola'))

True
endswith
Regresa verdadero si una cadena termina con cierta subcadena
print('hola mundo'.endswith('mundo'))

True
join
Concatena cadenas en una lista de cadenas, usando la cadena actual como separador
print(','.join(['hola', 'mundo', 'mundial']))
hola,mundo,mundial

Si necesitas modificar caracteres de una cadena, una forma simple y eficiente de hacerlo es primero convertir la cadena a lista, mediante la función list y luego de las modificaciones regresar a cadena con la función join usando la cadena vacía como separador

```
s = 'hola mundo'
ls = list(s)
print(ls)
ls[0] = 'H'
s = ''.join(ls)
print(s)
```

Insert
Al inicio
Cuidado: insert permite posiciones inválidas (mayores a la longitud o menores a 0)

```
l = [2, 3, 4]
l.insert(0, 1)
print(l)
```

R= [1, 2, 3, 4]

Al final
Usar método append de lista

```
l = [1, 2, 3]
l.append(4)
print(l)
```

R=[1, 2, 3, 4]

En cualquier posición
Con el método insert

```
l = [1, 2, 4]
l.insert(2, 3)
print(l)
```

[1, 2, 3, 4]

Remplazar elementos
Se puede con asignación directa
Es una operación in-place

```
l = [0, 2, 3]
l[0] = 1
print(l)
```

[1, 2, 3]
Borrar elementos
Operación in-place
Se utiliza la función general del que funciona para varias estructuras de datos de Python

```
l = [0, 1, 2, 3]
del(l[0])
print(l)
```

[1, 2, 3]

Obtener sublistas
Es una operación no mutable
Usando rebanadas (como se vio en el tema anterior)
Las rebanadas regresan nueva memoria (por eso son no mutables)

```
l = [1, 2, 3, 4]
print(l[:-1]) # todos menos último
print(l[1:]) # todos menos primero
print(l[1:-1]) # sin primero y último
```

[1, 2, 3]
[2, 3, 4]
[2, 3]

Orden ascendente

```
l = [44, 11, 7, 22]
l2 = sorted(l)
l.sort()
print(l)
print(l2)
```

[7, 11, 22, 44]
[7, 11, 22, 44]

Orden descendente
Se logra con el parámetro nombrado (keyword) reverse

```
l = [44, 11, 7, 22]
l2 = sorted(l, reverse=True)
l.sort(reverse=True)
print(l)
print(l2)
```

[44, 22, 11, 7]
[44, 22, 11, 7]

PILAS

Push: agrega un elemento al tope

Pop: saca y regresa el elemento al tope

Peek: sólo regresa el valor del elemento del tope sin sacarlo

```
pila = []
pila.append(1) # equivalente de push
pila.append(2)
pila.append(3)

tope = pila[-1] # equivalente de peek
print(tope)

tope = pila.pop()
print(tope)
print(pila)
```

COLAS

append: agrega un elemento al final (igual que en una lista)
shift: saca el elemento del frente y lo regresa
peek: sólo regresa el valor del elemento del frente sin sacarlo

```
print("¿Sigue", [0, 1, 1, 2, 3, 5], "la sucesión de Fibonacci?", end=" R: ")
print("No lo sé", "No me importa", sep=" y ")

print("F", "i", "b", "o", "n", "a", "c", "c", "i", sep="", end=": ")
print(0, 1, 1, 2, 3, 5, sep=", ", end=", ...")
```

```
¿Sigue [0, 1, 1, 2, 3, 5] la sucesión de Fibonacci? R: No lo sé y No me importa
Fibonacci: 0, 1, 1, 2, 3, 5, ...
```

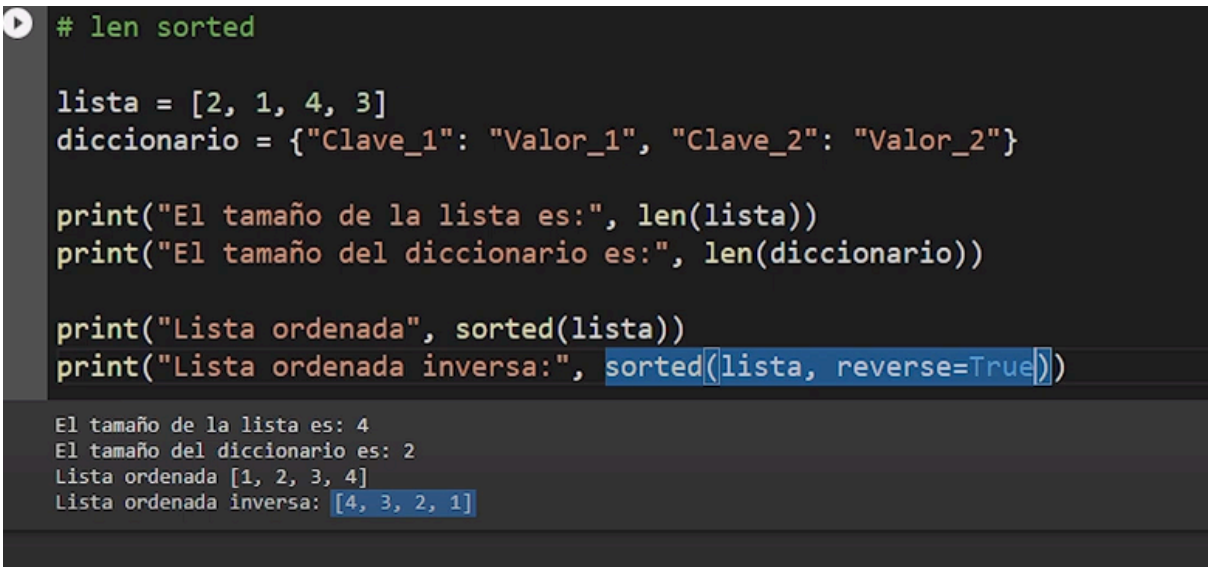
```
# len sorted

lista = [2, 1, 4, 3]
diccionario = {"Clave_1": "Valor_1", "Clave_2": "Valor_2"}

print("El tamaño de la lista es:", len(lista))
print("El tamaño del diccionario es:", len(diccionario))

print("Lista ordenada", sorted(lista))
```

```
El tamaño de la lista es: 4
El tamaño del diccionario es: 2
Lista ordenada [1, 2, 3, 4]
```



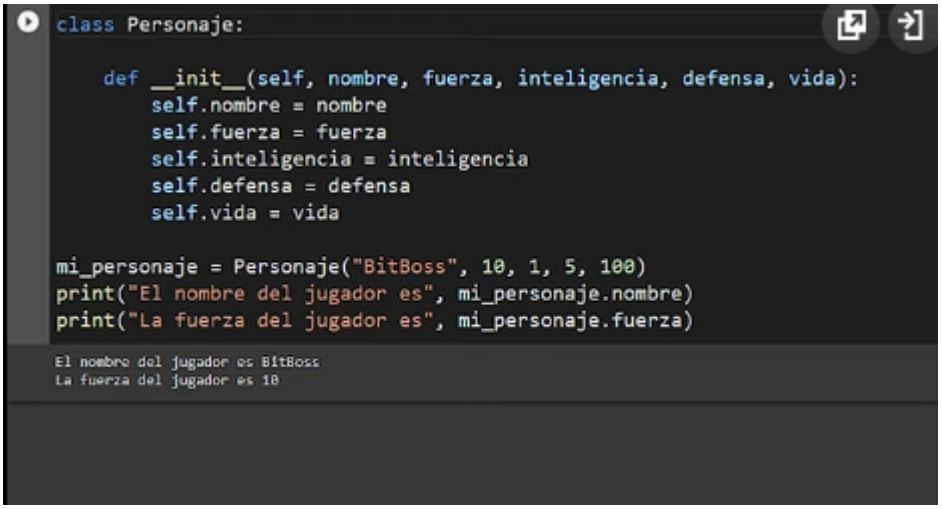
```
# len sorted

lista = [2, 1, 4, 3]
diccionario = {"Clave_1": "Valor_1", "Clave_2": "Valor_2"}

print("El tamaño de la lista es:", len(lista))
print("El tamaño del diccionario es:", len(diccionario))

print("Lista ordenada", sorted(lista))
print("Lista ordenada inversa:", sorted(lista, reverse=True))
```

```
El tamaño de la lista es: 4
El tamaño del diccionario es: 2
Lista ordenada [1, 2, 3, 4]
Lista ordenada inversa: [4, 3, 2, 1]
```



```
class Personaje:

    def __init__(self, nombre, fuerza, inteligencia, defensa, vida):
        self.nombre = nombre
        self.fuerza = fuerza
        self.inteligencia = inteligencia
        self.defensa = defensa
        self.vida = vida

mi_personaje = Personaje("BitBoss", 10, 1, 5, 100)
print("El nombre del jugador es", mi_personaje.nombre)
print("La fuerza del jugador es", mi_personaje.fuerza)
```

```
El nombre del jugador es BitBoss
La fuerza del jugador es 10
```

signo mayor >

SIGNO MENOR <

Función divmod()

`divmod( )` es una función incorporada en Python 3, que devuelve el cociente y el resto al dividir el número a por el número b. Toma dos números como argumentos a & b. El argumento no puede ser un número complejo.

Código de ejemplo:

```
print(divmod(5,2)) # muestra (2,1)
print(divmod(13.5,2.5)) # muestra (5.0, 1.0)
q,r = divmod(13.5,2.5)  # Asigna q=cociente & r=resto
print(q) # muestra 5.0 porque math.floor(13.5/2.5) = 5.0

print(r) # muestra 1.0 porque (13.5 % 2.5) = 1.0
```

Función Hex(x)

`hex(x)` es una función incorporada en Python 3 para convertir un número entero en una cadena hexadecimal en minúscula con el prefijo "0x"

Código de ejemplo:

```
print(hex(16))     # muestra  0x10
print(hex(-298))   # muestra -0x12a

print(hex(543))    # muestra  0x21f
```

Función Ord

`ord()` es una función incorporada en Python 3, para convertir la cadena que representa un carácter Unicode en un entero que representa el código Unicode del carácter.

Ejemplos:

```
>>> ord('d')
100
>>> ord('1')

49
```

Función chr

`chr()` es una función incorporada en Python 3, para convertir el número entero que representa el código Unicode en una cadena que representa un carácter correspondiente.

ABS

se utiliza para calcular el valor absoluto de un número. El valor absoluto de un número es su distancia respecto al cero en la recta numérica, es decir, siempre es un número positivo o cero.

EXAMPLE:

a=-42

print(abs(a))

ALL

busca que todo sea verdadero, si encuentra un falso en todo el proceso marca como falso

lista = [True, True, False, True]

print(all(lista)) # False, porque hay un False en la lista

tupla = (1, 2, 3, 4)print(all(tupla)) # True, todos los elementos son considerados verdaderos (truthy)

ANY

si hay un valor como verdadero marca verdadero

lista = [False, True, False, True]

print(any(lista)) # True, porque al menos hay un True en la lista

lista_vacia = []

print(any(lista_vacia)) # False, porque no hay elementos que evaluar

ASCII

devuelve el ascii de una cadena

print(ascii('Café')) # 'Caf\xe9', muestra la representación ASCII usando secuencias de escape

BIN

da el valor que ingresaste en binario

print(bin(10)) # '0b1010', representa el número 10 en binario

BOOL

sirve para ver si es verdadero o falso

print(bool(True))   # True

print(bool(1))   # True, cualquier número distinto de cero es verdadero

print(bool(False))   # False

print(bool(0))   # False, cero entero es falso

DIVMOD

toma dos números y devuelve una tupla que contiene el cociente y el residuo de la división entre esos dos números

resultado = divmod(11, 3)

print(resultado) # Output: (3, 2)

El cociente de dividir 11 entre 3 es 3.

El residuo de dividir 11 entre 3 es 2.

ENUMERATE

utiliza para agregar un contador a un iterable y devolverlo como un objeto enumerado

lista = ['a', 'b', 'c', 'd']

for indice, valor in enumerate(lista):

print(f'Índice: {indice}, Valor: {valor}')

Índice: 0, Valor: a

Índice: 1, Valor: b

Índice: 2, Valor: c

Índice: 3, Valor: d

FILTER

se utiliza para filtrar elementos de un iterable (como una lista, tupla, conjunto, etc.) basado en una función de filtro, que devuelve `True` o `False` para cada elemento del iterable.

Definir una función de filtro

def es_positivo(numero):

return numero > 0

```
# Crear una lista de números

numeros = [-2, -1, 0, 1, 2, 3, 4, 5]



# Filtrar los números positivos usando filter()

numeros_positivos = list(filter(es_positivo, numeros))



print(numeros_positivos)  # Output: [1, 2, 3, 4, 5]



# Filtrar números pares usando una función lambda

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



numeros_pares = list(filter(lambda x: x % 2 == 0, numeros))



print(numeros_pares)  # Output: [2, 4, 6, 8, 10]
```

FROZEMSET

soportan operaciones de conjunto como intersección, unión, diferencia y comprobaciones de subconjunto, al igual que los conjuntos normales. Sin embargo, dado que son inmutables, no admiten métodos que modifiquen el conjunto, como `add()`, `remove()`, o `discard()`.

```
fset1 = frozenset([1, 2, 3])

fset2 = frozenset([2, 3, 4])



# Intersección

intersection = fset1 & fset2

print(intersection)  # Output: frozenset({2, 3})



# Unión

union = fset1 | fset2

print(union)  # Output: frozenset({1, 2, 3, 4})



# Diferencia

difference = fset1 - fset2

print(difference)  # Output: frozenset({1})
```

GETATTR

integrada que se utiliza para obtener el valor de un atributo de un objeto dado su nombre. Permite acceder dinámicamente a los atributos de un objeto utilizando una cadena que especifica el nombre del atributo. Aquí te explico cómo funciona y cómo se utiliza `getattr()`:

```
class Persona:

    def __init__(self, nombre, edad):

        self.nombre = nombre

        self.edad = edad



# Crear un objeto Persona

persona = Persona("Alice", 30)



# Obtener dinámicamente el valor del atributo 'nombre'

nombre_persona = getattr(persona, 'nombre')

print(nombre_persona)  # Output: Alice



# Obtener dinámicamente el valor del atributo 'edad'

edad_persona = getattr(persona, 'edad')
```

```
print(edad_persona)  # Output: 30
```

ID

saber si son mismo o no a pesar de tener el mismo contenido pueden no ser el mismo

```
# Comparación de identificadores de objetos

a = [1, 2, 3]

b = a

c = [1, 2, 3]



print(id(a))  # Identificador de 'a'

print(id(b))  # Identificador de 'b' (mismo que 'a' porque 'b' apunta al mismo objeto)

print(id(c))  # Identificador de 'c' (diferente de 'a' y 'b' porque 'c' apunta a un objeto diferente)
```

MAP

es una función integrada que se utiliza para aplicar una función a cada elemento de uno o más iterables (como listas, tuplas, etc.) y devuelve un iterador que produce los resultados de aplicar la función a cada elemento.

```
# Definir una función que duplica un número

def duplicar(numero):

    return numero * 2



# Aplicar la función a cada elemento de una lista usando map()

numeros = [1, 2, 3, 4, 5]

resultado = map(duplicar, numeros)



# Convertir el iterador a una lista para ver los resultados

lista_resultado = list(resultado)

print(lista_resultado)  # Output: [2, 4, 6, 8, 10]
```

OCT

obtener base octal

```
# Convertir un número entero a octal

numero = 42

octal = oct(numero)
```

```
print(octal)  # Output: '0o52'
```

ORD

una función integrada que proporciona una forma de obtener el valor Unicode de un carácter específico, permitiendo trabajar con caracteres individuales de manera eficiente en aplicaciones que manejan texto y cadenas de caracteres.

```
# Obtener el valor Unicode de un carácter

valor_unicode = ord('A')



print(valor_unicode)  # Output: 65
```

POW

saber potencia

```
# Calcular 2 elevado a la potencia 3

resultado = pow(2, 3)



print(resultado)  # Output: 8
```

REVERSED

integrada que se utiliza para obtener un iterador que recorre una secuencia en orden inverso.

```
# Iterar sobre una tupla en orden inverso

mi_tupla = (10, 20, 30, 40, 50)

for numero in reversed(mi_tupla):

    print(numero)
```

ROUND

redondea un numero

```
# Redondear un número decimal a dos decimales

numero2 = 3.14159

resultado2 = round(numero2, 2(o al que se quiera))



print(resultado2)  # Output: 3.14
```

SET

cualquier cosa a set

```
# Crear un conjunto a partir de una lista

lista = [1, 2, 3, 4, 5]
```

```
conjunto_desde_lista = set(lista)



# Crear un conjunto a partir de una cadena

cadena = "hola"

conjunto_desde_cadena = set(cadena)
```

SLICE

para obtener subcadenas

```
slice(start, stop, step)

# Crear un objeto slice con un paso de 2 para seleccionar elementos alternos

mi_slice_step = slice(0, 6, 2)



# Aplicar el slice a una cadena

mi_cadena = "Python es genial"

resultado_step = mi_cadena[mi_slice_step]



print(resultado_step)  # Output: 'Pto'
```

TUPLE

convierte lo que sea a tupla

```
# Convertir una lista en una tupla

mi_lista = [10, 20, 30]

tupla_desde_lista = tuple(mi_lista)



# Convertir una cadena en una tupla de caracteres

cadena = "Python"

tupla_desde_cadena = tuple(cadena)
```

ZIP

combina listas

```
# Ejemplo con listas

numeros = [1, 2, 3]

letras = ['a', 'b', 'c']



resultado = zip(numeros, letras)
```

```
# Convertir el iterador zip a una lista de tuplas

lista_resultado = list(resultado)



print(lista_resultado)

# Output: [(1, 'a'), (2, 'b'), (3, 'c')]
```

ADD

agegar cosas a conjuntos

```
# Crear un conjunto vacío

my_set = set()



# Agregar elementos al conjunto usando el método add()

my_set.add(1)

my_set.add(2)

my_set.add(3)



print(my_set)  # Output: {1, 2, 3}
```

```
# Intentar agregar un elemento que ya existe
my_set.add(2)

print(my_set)  # El conjunto sigue siendo {1, 2, 3}, ya que el 2 ya estaba presente
```

DISCARD

```
# Crear un conjunto
frutas = {"manzana", "plátano", "uva", "naranja"}

# Eliminar un elemento existente con discard()
frutas.discard("uva")
print(frutas)  # Output: {'manzana', 'plátano', 'naranja'}

# Intentar eliminar un elemento que no está presente
frutas.discard("pera")
print(frutas)  # El conjunto sigue siendo {'manzana', 'plátano', 'naranja'}
```

LSTRIP

`str` en Python y se utiliza para eliminar caracteres desde el inicio (izquierda) de una cadena hasta que se encuentra un carácter que no esté incluido en los argumentos.

```
a=input("")

g=a.lstrip("0")

print(len(g))
```

ENTRADA:032

SALIDAA:2

POP

eliminar objeto en zona marcada

```
lista = [1, 2, 3, 4, 5]

elemento = lista.pop()

print(elemento)  # Salida: 5

print(lista)     # Salida: [1, 2, 3, 4]
```

UPPER

convierte TODO A MAYUSCULAS

TITLE

INICIALES A MAYUSCULAS

```
cadena = "hola, mundo. este es un ejemplo."

cadena_iniciales_mayusculas = cadena.title()

print(cadena_iniciales_mayusculas)  # Salida: Hola, Mundo. Este Es Un Ejemplo.
```