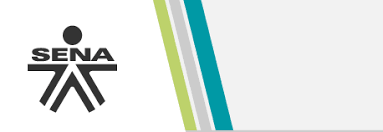
****

**BACKEND-PYTHON – PROGRAMACIÓN DE SOFTWARE**

**ACTIVIDAD 1: ESTRUCTURA DE DATOS.**

**INTEGRANTES**

**TANIA CAROLINA MURILLO LOPEZ**

**FICHA:2502640**

**PROGRAMACIÓN DE SOFTWARE**

**SENA**

**2022**

**CENTRO DE GESTIÓN DE MERCADOS LOGÍSTICA Y TECNOLOGÍAS DE LA INFORMACIÓN.**

**TABLA DE CONTENIDO**

* **INTRODUCCIÓN**

[**1. Más sobre listas**](#_heading=h.9rvfqx8w1lxa) **3**

[2. Usando listas como pilas](#_heading=h.lv20kr5wjw7v) 6

[3. Usando listas como colas](#_heading=h.svcql8tahiba) 6

[4. Compresión de listas](#_heading=h.k70i7jgcj4xr) 7

[5. listas por Comprensión animadas](#_heading=h.b9b2ruljgfl) 7

[6. la instrucción del](#_heading=h.oaxpu42fb5nq) 8

[7. tuplas y secuencias](#_heading=h.aj21cwxb3ys7) 8

[8. Conjuntos](#_heading=h.ln0ryzi6uiu) 9

[9. Diccionarios](#_heading=h.4kcctz4n7ays) 9

[10.Técnicas de Iteración](#_heading=h.4na4jdvo6goo) 10

[11.Más acerca de condiciones](#_heading=h.4v20tfpobr7w) 12

[12.Comparando secuencias y otros tipos](#_heading=h.nhs3bcilmsz) 12

* **WEBGRAFÍA Y CIBERGRAFÍA: REPLIT**

**ESTRUCTURA DE DATOS**

**TALLER No. 1**

Las estructuras de datos en Python pueden entenderse como tipos de datos compuestos porque podemos almacenar estructuras e información completas en una sola variable, asu vez puede tener diferentes propiedades y funciones por otra parte sus estructuras de datos más comunes son listas, tablas y diccionarios, aunque tienen un nombre diferente, en otros lenguajes son arreglos o vectores, matrices y arreglos indexados.

## 1. Más sobre listas

métodos de los objetos lista:

**-list.append(x):**

Agregue un elemento al final de la lista. Equivalente a a[len(a):] = [x].

**ejemplo:**

mi\_lista = [435, 324, 275, 567, 123]

mi\_lista.append(456)

print (mi\_lista)

RTA://

[435, 324, 275, 567, 123, 456]

**-list.extend(iterable):**

Expande la lista agregando todos los elementos del iterable a la lista. Equivale a a[len(a):] = iterable.

**ejemplo:**

numeros = [5.6, 7.44, 6.75, 4.56, 2.3]

valores\_nuevos = [2.3, 9.6, 4.564, 7.56]

numeros.extend (valores\_nuevos)

print(numeros)

RTA://

[5.6, 7.44, 6.75, 4.56, 2.3, 2.3, 9.6, 4.564, 7.56]

-**list.insert(i, x):**

Inserta un elemento en la posición dada. El primer argumento es el índice antes del elemento que se va a insertar, por lo que a.insert(0, x) se inserta al principio de la lista, a.insert(len(a), x) es equivalente a a.append(x) ) ).

Método: insert(posición, "nuevo elemento")

**ejemplo:**

nombres\_masculinos= [ 'Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose',

'Jose', 'Gerardo']

nombres\_masculinos.insert(0, "Ricky")

print (nombres\_masculinos)

RTA://

['Ricky', 'Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose', 'Jose', 'Gerardo']

**-list.remove(x):**

Elimina el primer elemento de la lista con valor x. Lanza ValueError si no existe tal elemento.

**ejemplo:**

prime\_numbers = ['2', '3', '5', '7', '9', '11']

prime\_numbers.remove('9')

print( prime\_numbers)

RTA://

['2', '3', '5', '7', '11']

**-list.pop([i]):**

Elimina el elemento en la posición especificada de la lista y lo devuelve. Si no se especifica ningún índice, a.pop() aparecerá y devolverá el último elemento de la lista. (Los corchetes alrededor de i en la firma del método indican que el parámetro es opcional, no que los corchetes deban estar allí. Verá esta notación con bastante frecuencia en la referencia de la biblioteca de Python).

**ejemplo:**

nombres\_masculinos = [ 'Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose',

'Jose', 'Gerardo']

nombres\_masculinos.pop (1)

print (nombres\_masculinos)

RTA://

['Alvaro', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose', 'Jose', 'Gerardo']

**-list.clear():**

Eliminar todos los elementos de la lista. Coincide con la parte a[:].

**ejemplo:**

list = [{1, 2, 3, 4}, ['1.1', '2.2']]

list.clear()

print(list)

RTA://

[ ]

**-list.index(x[, start[, end]]):**

El valor devuelto es igual al índice basado en cero del primer elemento de x. Si no existe tal elemento, se lanza una excepción ValueError, Los argumentos de inicio y final opcionales se interpretan como una notación de división utilizada para limitar la búsqueda a un segmento específico de la lista. El índice devuelto se calcula en relación con el inicio de toda la secuencia, no en relación con el parámetro de inicio.

list.index(elemento, inicio, fin)

**ejemplo:**

animals = ['cat', 'dog', 'rabbit', 'horse']

index = animals.index('rabbit')

print(index)

RTA://

2

**-list.count(x):**

Devuelve el número de veces que x aparece en la lista.

**ejemplo:**

list1 = [1, 1, 1, 2, 3, 2, 1]

print(list1.count(1))

list2 = ['a', 'a', 'a', 'b', 'b', 'a', 'c', 'b']

print(list2.count('b'))

RTA://

4

3

**-list.sort(\*, key=None, reverse=False):**

Ordena los elementos de la lista en su lugar (los parámetros se pueden usar para ajustar el orden de la lista, consulte sorted() para obtener una explicación).

False significa que la lista será ordenada de forma ascendente.

True significa que la lista será ordenada de forma descendente.

**ejemplo:**

mi\_lista = [6, 3, 8, 2, 7, 3, 9]

mi\_lista.sort()

print(mi\_lista)

RTA://

[2, 3, 3, 6, 7, 8, 9]

# ¡Ordenada!

c = ["A", "Z", "D", "T", "U"]

c.sort(reverse=True)

print(c)

RTA://

['Z', 'U', 'T', 'D', 'A']

**-list.reverse():**

Elementos de la lista invertida.

**ejemplo:**

capitals = ['Oslo', 'Skopje', 'Riga', 'Madrid']

capitals.reverse()

print(capitals)

RTA://

['Madrid', 'Riga', 'Skopje', 'Oslo']

**-list.copy():**

Devuelve una copia superficial de la lista. Se refiere a [:].

**ejemplo:**

fruits\_spring=['carrots', 'kiwi', 'grapes', 'cherry']

fruits\_summer=fruits\_spring.copy()

print(fruits\_summer)

RTA://

['carrots', 'kiwi', 'grapes', 'cherry']

### 2. Usando listas como pilas

el último elemento añadido es el primer elemento retirado

**ejemplo:**

lisT = [3, 4, 5]

lisT.append(6)

lisT.append(7)

print(lisT)

RTA://

[3, 4, 5, 6, 7]

Para retirar un elemento de la cima de la pila, utiliza pop()

lisT.pop()

RTA://[3, 4, 5, 6]

lisT.pop()

RTA://[3, 4, 5]

### 3. Usando listas como colas

El primer elemento en la lista será el primero en ser eliminado.

-utiliza collections.deque el cual fue diseñado para añadir y quitar de ambas puntas de forma rápida.

**ejemplo:**

from collections import deque

liscola = deque(["1", "2", "3"])

liscola.append ("4")

liscola.append("5")

liscola.popleft()

#se elimina 1

liscola.popleft()

#se elimina 2

print (liscola)

RTA://

deque(['3', '4', '5'])

### 4. Compresión de listas

La comprensión de listas es una construcción en Python que nos permite crear listas a partir de otras listas.

**ejemplo1:**

lista = [letra for letra in 'casa']

print(lista)

RTA://

['c', 'a', 's', 'a']

**ejemplo2:**

lista = [numero\*\*2 for numero in range(0,11)]

print(lista)

RTA://

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

### 5. listas por Comprensión animadas

La comprensión de listas es una construcción que nos permite crear listas a partir de otras listas

**ejemplo:**

matrix = [

[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

]

matrix = [[row[i] for row in matrix] for i in range(4)]

#otra forma

matrix =list(zip(\*matrix))

print (matrix)

RTA://

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

### 6. la instrucción del

La instrucción del también puede usarse para quitar secciones de una lista o vaciar la lista completa

**ejemplo:**

a = [-1, 5, 5.25, 3, 3, 34.5]

del a[0]

print (a)

RTA://

[5, 5.25, 3, 3, 34.5]

del a[2:4]

RTA://[-1, 5, 3, 34.5]

del a[:]

RTA:// [ ]

del puede usarse también para eliminar variables

### 7. tuplas y secuencias

Una tupla consiste de un número de valores separados por comas

**ejemplo:**

s = 12, 14, 'wiski'

u = s, (1,2,3,4,5)

print(u)

RTA://

((12, 14, 'wiski'), (1, 2, 3, 4, 5))

las tuplas se pueden crear vacías al representarlas con un paréntesis

La función len() devuelve el número de elementos de una tupla:

empty = ()

singleton = ('hello', 'f' )

empty =len(empty)

singleton=len(singleton)

print (empty)

RTA:// 0

print (singleton)

RTA:// 2

### 8. Conjuntos

La colección no ordenada y sin elementos repetidos,también permite la unión, intersección, diferencia y diferencia simétrica. Puede crear un conjunto vacío con la función set(). Para crear una colección, usamos llaves {}.

**ejemplo:**

s = {1, 2, 3, 4}

print(s)

RTA://1, 2, 3, 4

crear conjuntos de un solo elemento

a = set('uva')

b = set('alakazam')

para revisar los elementos únicos en un conjunto

print(a)

para encontrar las letras que se encuentran en a pero no en b

print(a - b)

para encontrar las letras que se encuentran en a, b o los dos

print(a | b )

para encontrar las letras presentes en a y b

print(a&b)

para encontrar las letras en a o b, pero no en las dos

print(a ^ b)

los conjuntos también permiten su comprensión, el siguiente código nos muestra las letras presentes en c que no son abc

c = {x for x in 'abracadabra' if x not in 'abc'}

### 9. Diccionarios

Los diccionarios de Python nos permiten almacenar una serie de asociaciones entre dos conjuntos de elementos llamados keys and values (claves y valores).

Todos los elementos en el diccionario se encuentran encerrados en un par de corchetes {}.

También es posible borrar un par clave:valor con del. Si usás una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde.

Para comprobar si una clave está en el diccionario usa la palabra clave in.

**ejemplo:**

s = {'ana':1, 'uva':2, }

s ['vino'] = 3

print(s)

RTA://{'ana': 1, 'uva': 2, 'vino': 3}

#averiguar clave

print (s['ana'])

RTA:// 1

Utilizando del para eliminar

del s['uva']

print(s)

RTA://{'ana': 1, 'vino': 3}

-Ejecutando list(s) en un diccionario retornará una lista con todas las claves usadas en el diccionario.

print (list(s))

RTA://['ana', 'uva', 'vino']

-si deseas que esté ordenada simplemente usa sorted(s) en su lugar).

-Para comprobar si una clave está en el diccionario usa la palabra clave in.

-El constructor dict() crea un diccionario directamente desde secuencias de pares clave-valor:

print (dict(s))

RTA://{'ana': 1, 'uva': 2, 'vino': 3}

Además, las comprensiones de diccionarios se pueden usar para crear diccionarios desde expresiones arbitrarias de clave y valor.

s{x: x\*\*2 for x in (2, 4, 6)}

print( s)

### 10.Técnicas de Iteración

Al iterar sobre un diccionario, puede usar el método items() para obtener ambas claves y sus valores correspondientes.

**ejemplo:**

s = {'a':1, 'b':2, 'c':3, 'b':4, }

for k,v in s.items():

print (k,v)

RTA://

a 1

b 4

c 3

Cuando se itera sobre una secuencia, se puede obtener el índice de posición junto a su valor correspondiente usando la función enumerate().

**ejemplo:**

for u, v in enumerate(['yo', 'soy', 'tu']):

print(u, v)

RTA://

0 yo

1 soy

2 tu

Para iterar sobre dos o más secuencias al mismo tiempo, los valores pueden emparejarse con la función zip().

**ejemplo:**

preguntas = ['nombre', 'búsqueda', 'color favorito']

respuestas = ['lancelot', 'el santo grial', 'azul']

for p3, r3 in zip(preguntas, respuestas):

print('¿Cuál es tu {0}? Es {1}'.format(p3, r3))

RTA://

¿Cuál es tu nombre? Es lancelot

¿Cuál es tu búsqueda? Es el santo grial

¿Cuál es tu color favorito? Es azul

Para iterar sobre una secuencia en orden inverso, se especifica primero la secuencia al derecho y luego se llama a la función reversed().

**ejemplo:**

for i in reversed(range(1, 10, 2)):

print(i)

RTA://

9

7

5

3

1

iterar sobre una secuencia ordenada, se utiliza la función sorted() la cual retorna una nueva lista ordenada dejando a la original intacta.

**ejemplo:**

usar = ['1', '2', '1', '3', '2', '4']

for i in sorted(usar):

print(i)

RTA://

1

1

2

2

3

4

El uso de set() en una secuencia elimina los elementos duplicados. El uso de sorted() en combinación con set() sobre una secuencia es una forma idiomática de recorrer elementos únicos de la secuencia en orden ordenado.

**ejemplo:**

usar = ['1', '2', '1', '3', '2', '4']

for f in sorted(set(usar)):

print(f)

RTA://

1

2

3

4

A veces uno intenta cambiar una lista mientras la está iterando; sin embargo, a menudo es más simple y seguro crear una nueva lista:

**ejemplo:**

import math

usar = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]

data = []

for value in usar:

if not math.isnan(value):

data.append(value)

print(data)

RTA://

[56.2, 51.7, 55.3, 52.5, 47.8]

### 11.Más acerca de condiciones

Las comparaciones pueden combinarse mediante los operadores booleanos and y or

**ejemplo:**

cadena1, cadena2, cadena3 = '', 'sol', 'luna'

non\_null = cadena1 or cadena2 or cadena3

print(non\_null)

RTA://

sol

### 12.Comparando secuencias y otros tipos

Si dos ítems a comparar son ambas secuencias del mismo tipo, la comparación lexicográfica es recursiva.

El orden lexicográfico(orden de diccionario) de los strings utiliza el punto de código Unicode para ordenar caracteres individuales.

**ejemplo:**

Entrada: "hola programa Python, ¿cómo estás?"

Salida: son

Hola

Cómo

programa

python

Uds

Entrada: "A los programadores les encantan los algoritmos"

Salida: codificadores

algoritmos

ama

la

* **WEBGRAFÍA**

**-**https://www.freecodecamp.org/espanol/news/python-lista-append-como-agregar-elementos-a-una-lista-explicado-con-ejemplos/

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