# ACTIVIDAD #8

## TALLER DE CONOCIMIENTO --- ESTRUCTURA DE DATOS

1. **¿Qué son las estructuras de datos?**

**R/:** Las **estructuras de datos en programación** son un modo de representar información en una computadora, aunque, además, cuentan con un comportamiento interno. ¿Qué significa? Que se rige por determinadas reglas/restricciones que han sido dadas por la forma en que está construida internamente.

Las estructuras de datos son aquellas que nos permiten, como desarrolladores, organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.

Ya sean las más utilizadas comúnmente -como las variables, arrays, conjuntos o clases- o las diseñadas para un propósito específico -árboles, grafos, tablas, etc.-, una estructura de datos nos permite trabajar en un algo nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla.

1. **Tipos de Estructuras de datos**

**R/:** debemos diferenciar entre estructura de dato estática y estructura de dato dinámica

Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa, mientras que una estructura de datos dinámica es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.

Cada tipo de estructura dependerá del tipo de aplicación que se requiera. Una típica dentro de las estructuras de datos estáticas son los arrays:

Arrays

¿Qué es un array en programación? Un array es un tipo de dato estructurado que permite almacenar un conjunto de datos homogéneo y ordenado, es decir, todos ellos del mismo tipo y relacionados. Su condición de homogéneo, indica que sus elementos están compuestos por el mismo tipo de dato, y su condición de ordenado hace que se pueda identificar del primer al último elemento que lo compone.

Estructura de datos dinámicas

Por otro lado, vimos que en programación existen estructuras de datos dinámicas, es decir, una colección de elementos -nodos- que normalmente se utilizan para dejar asentados registros. A diferencia de un array que contiene espacio para almacenar un número fijo de elementos, una estructura dinámica de datos se amplía y contrae durante la ejecución del programa. Veamos algunos casos:

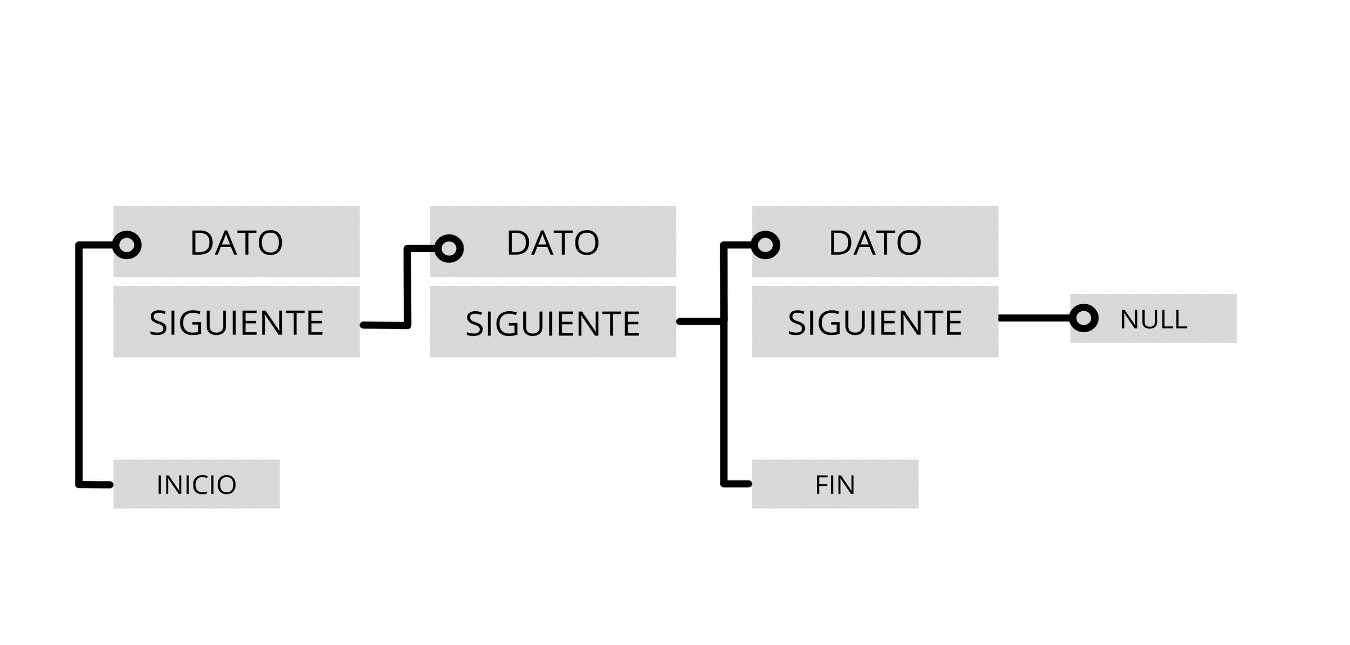
Estructura de datos lineales

Las estructuras de datos lineales son aquellas en las que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y predecesor, es decir, sus elementos están ubicados uno al lado del otro relacionados en forma lineal.

Hay tres tipos de estructuras de datos lineales:

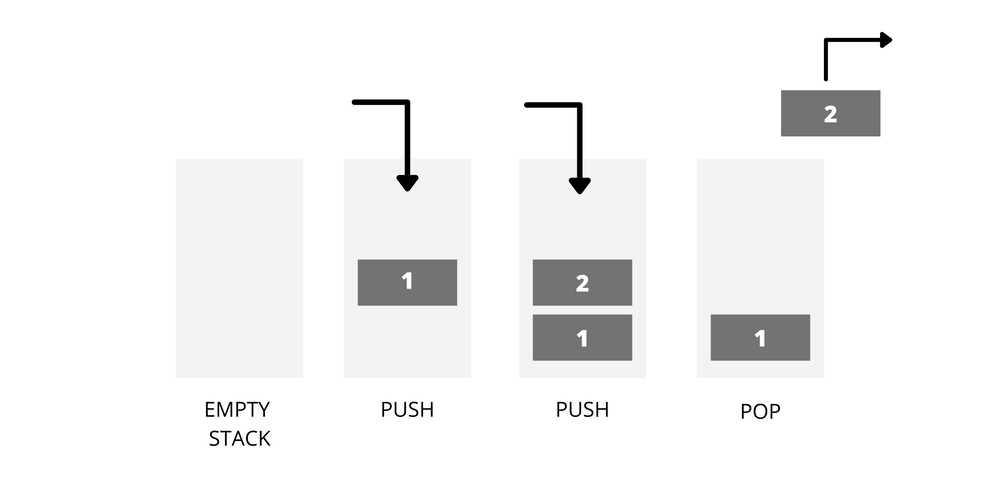
* Listas enlazadas
* Pilas
* Colas
* **Listas enlazadas**

En las estructuras de datos, las listas enlazadas se construyen con elementos que están ubicados en una secuencia. Aquí, cada elemento se conecta con el siguiente a través de un enlace que contiene la posición del siguiente elemento. De este modo, teniendo la referencia del principio de la lista podemos acceder a todos los elementos de la misma.



* **Pila**

La pila es un tipo especial de lista lineal dentro de las estructuras de datos dinámicas que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo LIFO (del inglés Last In, First Out, es decir, último en entrar, primero en salir). ¿Cómo funciona? A través de dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, desapilar (pop), que retira el último elemento apilado.



* **Estructura de datos no lineales**

Las estructuras de datos no lineales, también llamadas multienlazadas, son aquellas en las que cada elemento puede estar enlazado a cualquier otro componente. Es decir, cada elemento puede tener varios sucesores o varios predecesores.

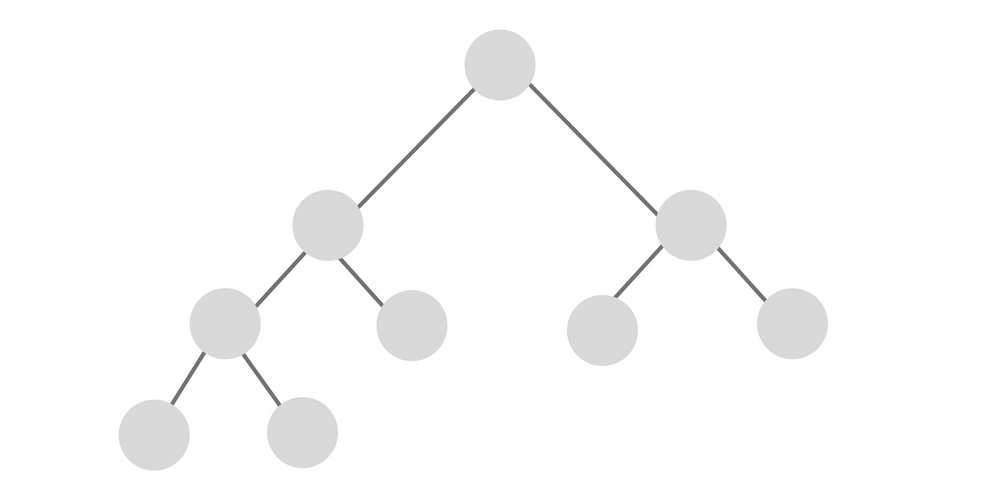
**Existen dos tipos:**

* Árboles
* Grafos
* Árboles

En estructura de datos, los árboles consisten en una estructura no lineal que se utiliza para representar datos con una relación jerárquica en la que cada elemento tiene un único antecesor y puede tener varios sucesores.

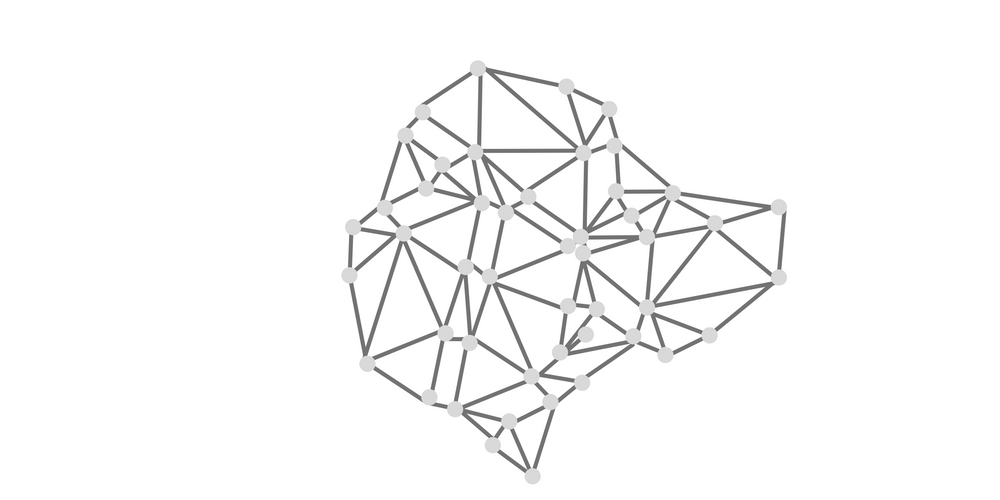
Los mismos se encuentran clasificados en:

* **árbol general:** un árbol donde cada elemento puede tener un número ilimitado de sub árboles y árboles binarios, que son una estructura de datos homogénea, dinámica y no lineal en donde a cada elemento le pueden seguir como máximo dos nodos.



* **Grafos**

Otro tipo de no lineal de estructura de datos en programación, son los grafos. Se trata de una estructura matemática formada por un conjunto de puntos —una estructura de datos— y un conjunto de líneas, cada una de las cuales une un punto a otro. Los puntos se llaman nodos o vértices del grafo y las líneas se llaman aristas o arcos.



1. **Estructuras de datos en Python**

**R/:**Las estructuras de datos en Python se pueden entender como un tipo de dato compuesto, debido a que en una misma variable podemos almacenar una estructura completa con información. Dichas estructuras, pueden tener diferentes características y funcionalidades. De hecho, existen múltiples tipos de estructuras de datos en Python

Las estructuras de datos más comunes en Python son las listas, las tablas y los diccionarios. Aunque tienen otro nombre, en otros lenguajes, son los arreglos o vectores, las matrices y los arreglos indexados, respectivamente. Son en esencia lo mismo, aunque como es habitual en Python, con varias facilidades y funcionalidades ya incluidas.

Las estructuras de datos han sido creadas para solucionar una gran variedad de problemáticas que no podrían ser resueltas con lo que conocemos hasta ahora, pues permiten agrupar fácilmente un conjunto de datos (normalmente relacionados) para operar fácilmente con ellos. Cosas como ordenar, agregar, eliminar, mostrar, recorrer, entre otras operaciones, son posibles y fáciles en Python.

1. Definir los tipos de datos en Python

**R/:**

* Listas
* Tuplas
* Sets
* Strings
* Diccionarios
* Listas
* **Las listas o arrays** en Python son estructuras de datos muy flexibles en las que podemos mezclar valores de varios tipos, o bien que sean de un solo tipo. Su declaración es sencilla y obedece un formato JSON estándar.

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

[1, 2, 3, 4]

>>> ["hola", "mundo"]

["hola", "mundo"]

>>> [0, 1.5, "hola"]

[0, 1.5, "hola"]

>>> [0, 1.5, "hola"]

[0, 1.5, "hola"]

Como se ven en estos ejemplos, podemos mezclar números enteros con cadenas e incluso números con puntos decimales.

Además una lista en Python puede contener en una o mas de sus posiciones otra lista de n dimensiones. formando así una lista ortogonal.

>>> a = [1, 2]

>>> b = [1.5, 2, a]

>>> b

[1.5, 2, [1, 2]]

Append y Extend

Las dos operaciones básicas en las listas son las de append y extend. La diferencia entre estas dos operaciones es que append agrega un elemento al final de la lista, mientras que extend es capaz de agregar otra lista al final de la lista. Veamos unos ejemplos:

– Append:

>>> lista = ['a','b']

>>> lista.append('c')

>>> stack

['a', 'b', 'c']

– Extend

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

>>> lista.extend(['d', 'e','f'])

>>> lista

['a', 'b', 'c', 'd', 'e', 'f']

Operaciones con listas

Ahora veamos todas las operaciones posibles con listas en Python:

index

Nos devuelve la posición en la lista del elemento indicado

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

>>> lista.index('b')

1

insert

Inserta un elemento en la lista según el indicie deseado.

#['b', 'c', 'b']

>>> lista.insert(2, 'a')

>>> lista

['b', 'c', 'a', 'b']

remove

De manera similar se puede remover un elemento de la lista.

#['b', 'c', 'a', 'b']

>>> lista.remove(2, 'a')

>>> lista

['b', 'c', 'b']

pop

Pop devuelve el ultimo elemento de la lista y posteriormente lo remueve de la misma.

#['b', 'c', 'b']

>>> lista.pop()

'b'

>>> lista

['b', 'c']

count

Devuelve la cuenta de elementos actualmente en la lista

#['b', 'c', 'b']

>>> lista.count('b')

2

Sort

Existen varias posibilidades de ordenar un arreglo según se indique en el método sort. Para ordenar en orden ascendente por default:

>>> mi\_lista.sort()

>>> mi lista

['a', 'b', 'b', 'c']

o bien en orden descendente.

>>> mi\_lista.sort(reverse=True)

>>> mi lista

['c', 'b', 'b', 'a']

O se puede simplemente invertir el orden de los elementos sin ordenarlos previamente

>>> my\_list = ['a', 'c' ,'b']

>>> my\_list.reverse()

>>> my\_list

['b', 'c', 'a']

Como verán la simplicidad siempre fue tomada en cuenta cuando diseñaron este lenguaje tan maravilloso.

* **Tuplas**

Las tuplas son como las listas, excepto que en este caso son inmutables. Una tupla consiste de varios valores separados por comas.

>>> a = (1, 2, 3)

>>> a[0]

1

Algunas funciones de las listas también funcionan en las tuplas, como len y la sintaxis para obtener porciones del arreglo o slicing

>>> len(a)

3

>>> a[1:]

2, 3

Una cosa importante que hay que notar con las tuplas, es que como los paréntesis también se usan en Python para agrupar expresiones, si quisiéramos crear una tupla con un solo valor simplemente agregamos una coma al final de esta manera:

>>> a = (1)

>> a

1

>>> b = (1,)

>>> b

(1,)

>>> b[0]

1

* **Sets**

Los sets en Python son listas sin un orden específico pero cuyos elementos son únicos, es decir, no existe la repetición. Y para definir uno hay que explícitamente indicar que queremos que la lista sea un set con la función con el mismo nombre,

>>> x = set([3, 1, 2, 1])

set([1, 2, 3])

A partir de Python 2.7, podemos usar llaves para definir sets

>>> x = {3, 1, 2, 1}

set([1, 2, 3])

Podemos agregar nuevos elementos con la función add

>>> x = set([1, 2, 3])

>>> x.add(4)

>>> x

set([1, 2, 3, 4])

Igual que en las listas, podemos revisar si hay un elemento en el set usando la función in

>>> x = set([1, 2, 3])

>>> 1 in x

True

>>> 5 in x

False

* **Strings**

Una de las cosas curiosas sobre Python, es que en muchas maneras podemos manipular strings de la misma manera como se manipulan las listas. Esto convierte a los strings en una especie de estructura de datos conveniente de usar.

Podemos usar len en un string

>>> len("abrakadabra")

11

Y encontrar partes o segmentos del string según nos parezca conveniente.

>>> a = "helloworld"

>>> a[1] #primer caracter

'e'

>>> a[-2] # penúltimo caracter

'l'

>>> a[1:5] #caracteres del uno al cinco

"ello"

>>> a[:5] #caracteres del primero al quinto

"hello"

>>> a[5:] #caracteres del quinto al ultimo

"world"

>>> a[-2:] # los últimos dos caracteres de la cadena

'ld'

>>> a[:-2] # toda la cadena menos dos caracteres

'hellowor'

>>> a[::-1] # revertir el orden de la cadena

'dlrowolleh'

así mismo podemos usar el operador in para buscar en la cadena

>>> 'hell' in 'hello'

True

>>> 'full' in 'hello'

False

>>> 'el' in 'hello'

True

Hay varios métodos muy útiles cuando se trata de manipular cadenas como split() el cual guarda la cadena en un arreglo según el carácter que indiquemos, o según los espacios en blanco si no especificamos ninguno.

>>> "hello world".split()

['hello', 'world']

>>> "a,b,c".split(',')

['a', 'b', 'c']

Si quisieramos volver a unir los elementos del arreglo en un string podemos hacer uso del metodo join

>>> " ".join(['hello', 'world'])

'hello world'

El metodo strip, remueve el caracter o la secuencia de caracteres de la cadena segun se lo indiquemos.

>>> ' hello world\n'.strip()

'hello world'

>>> 'abcdefgh'.strip('abdh')

'cdefg'

además Python soporta formatos en sus strings que nos recuerdan a aquel viejo y confiable preparedStatement

>>> a = 'hola'

>>> b = 'Python'

>>> "%s %s" % (a, b)

'hola Python'

>>> 'Capitulo %d: %s' % (1, 'Estructuras de datos')

'Capitulo 1: Estructuras de datos'

* **Diccionarios**

Un diccionario en Python actúa de manera similar a una lista, excepto que el índice de este no necesariamente tiene que ser un numero entero, y se asemejan bastante a un objeto JSON

>>> a = {'x': 1, 'y': 2, 'z': 3}

>>> a['x']

1

>>> a['z']

3

>>> b = {}

>>> b['x'] = 2

>>> b[2] = 'foo'

>>> b[(1, 2)] = 3

>>> b

{(1, 2): 3, 'x': 2, 2: 'foo'}

para borrar un elemento de un diccionario, utilizamos el comando del

>>> a = {'x': 1, 'y': 2, 'z': 3}

>>> del a['x']

>>> a

{'y': 2, 'z': 3}

El método keys devuelve todas las llaves en el diccionario. El método valué devuelve todos los valores dentro del diccionario, y el método ítems devuelve todos los pares llave-valor dentro del diccionario en cuestión

>>> a.keys()

['x', 'y', 'z']

>>> a.values()

[1, 2, 3]

>>> a.items()

[('x', 1), ('y', 2), ('z', 3)]

Podemos iterar fácilmente en un diccionario utilizando la notación for…in

>>> for key in a: print key

...

x

y

z

>>> for key, value in a.items(): print key, value

...

x 1

y 2

z 3

Podemos igual que en todas las estructuras anteriores, buscar dentro de un diccionario con la palabra in

>>> 'x' in a

True

>>> 'p' in a

False

>>> a.has\_key('x')

True

>>> a.has\_key('p')

False

Así mismo podemos obtener un ítem de la lista específicamente con el método get, y definir que regresar en caso el ítem no exista dentro del diccionario

>>> d = {'x': 1, 'y': 2, 'z': 3}

>>> d.get('x', 5)

1

>>> d.get('p', 5)

5

>>> d.setdefault('x', 0)

1

>>> d

{'x': 1, 'y': 2, 'z': 3}

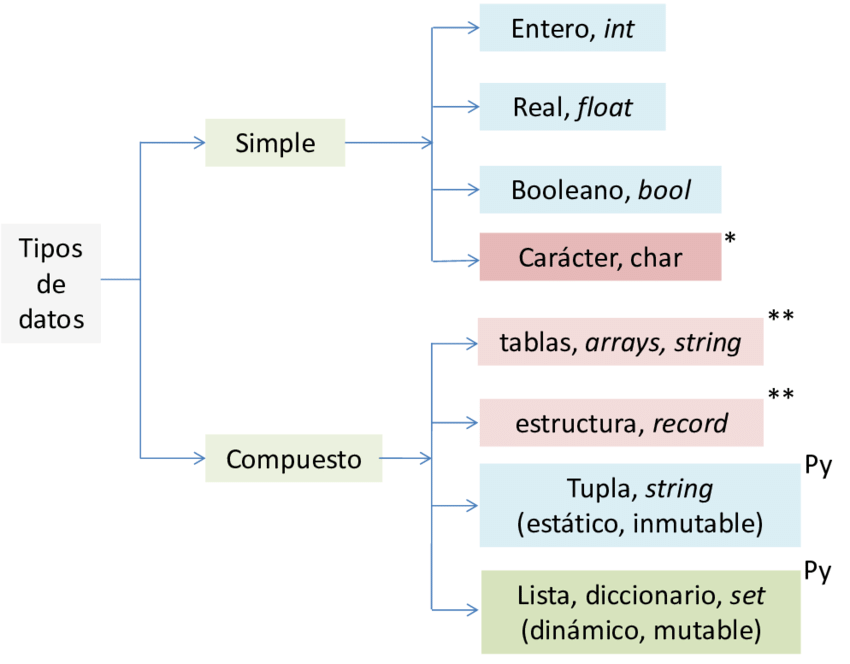
>>> d.setdefault('p', 0)

0

>>> d

{'y': 2, 'x': 1, 'z': 3, 'p': 0}

1. **Realice un grafico o dibujo de los tipos de datos**

****