Las estructuras de datos es una rama de las ciencias de la computación que estudia y aplica diferentes formas de organizar información dentro de una aplicación, para manipular, buscar e insertar estos datos de manera eficiente.

Entre las diferentes estructuras de datos podemos encontrar las siguientes:

## **Arrays**

Los arrays son una estructura que almacena los datos un elemento al lado del otro. En la mayoría de lenguajes de programación esta estructura de datos es de tamaño fijo y no puede guardar datos de diferentes tipos, es decir no puedo guardar valores numéricos y booleanos al mismo tiempo, aunque claramente hay excepciones, por ejemplo: JavaScript.

Es recomendable usar arrays cuando el acceso a estos datos se realiza de manera aleatoria, en caso contrario es recomendable usar las listas.

## **Listas enlazadas**

Las listas enlazadas son un tipo de estructura de datos similar a los arrays con la diferencia de que por defecto no tenemos por qué saber la cantidad de elementos que va a contener. Estas listas se componen de nodos los cuales tienen dos atributos: el primero es el ítem o elemento que va a contener este nodo y el segundo atributo es una referencia al siguiente elemento de la lista.

## **Pilas**

Las pilas son un tipo de listas que tienen la particularidad de sólo poder eliminar o insertar en la cima de la lista. A estas acciones se le conocen como apilar y des apilar y conlleva a que el último elemento que ingresa a la pila sea el primero en salir a lo cual se le conoce como LIFO (Last in First out).

## **Colas**

Esta estructura es otro tipo de lista que nos permite emular el comportamiento de una fila o cola de la vida real donde el primer elemento en ingresar a la fila es el primero en salir, lo que quiere decir que las inserciones (Encolar) se realizan al final y las extracciones (Desencolar) se realizan al frente de la cola, lo cual se conoce como FIFO (First in First out).

## **Arboles binarios**

Los árboles binarios son estructuras de datos que se componen de una nueva clase de nodo donde cada uno contiene un ítem o un valor, una referencia a un nodo que será el hijo izquierdo y otra referencia para el nodo derecho, una característica esencial de los árboles binarios es que la inserción de sus elementos se realiza siguiendo un criterio, si el ítem del nodo a insertar es menor a su nodo padre, la inserción se realiza por la izquierda y en caso contrario la inserción se realiza por

el lado derecho y como consecuencia de esto nuestro árbol siempre estará organizado de tal manera que los hijos izquierdos de cada nodo serán menores a él y los derechos serán mayores, lo cual nos permite realizar búsquedas muy eficientes

debido a la organización de esta información donde para buscar un elemento solo es necesario ir comparando el valor a buscar con el valor del nodo actual y así tomar la decisión si ir por la izquierda o por la derecha, ahorrándonos las búsquedas por el otro lado del árbol.

Espero que te haya gustado esta breve introducción a algunas de las estructuras de datos que existen.

(**Diagrama Nassi-Shneiderman**). Son estructuras que permiten representar de forma gráfica las [**sentencias**](https://www.glosarioit.com/Sentencia) que deberá ejecutar un [**algoritmo**](https://www.glosarioit.com/Algoritmo). Existe un **diagrama de Chapín** para cada una de las estructuras repetitivas y de selección conocidas y demás sentencias.  
Cuando se emprenda el desarrollo de un programa, con un cierto nivel de dificultad, siempre es recomendable la realización de un diagrama. De este modo, se podrán evitar o reconocer los errores fácilmente y permitirá orientarse sobre cómo crear el programa.

Los **diagramas de Chapín** no son los únicos desarrollados para este fin. Existen muchos otros en donde, básicamente varían los símbolos empleados para cada tarea, pero su finalidad sigue siendo la misma: ayudar al [**programador**](https://www.glosarioit.com/Programador) a diagramar el flujo del programa.  
El **diagrama de Chapín**, representa los distintos pasos del programa por medio de rectángulos uno debajo de otro. Además, también cuenta con otro tipo de figuras para la representación de decisiones o ciclos de iteración.

## **Entrando en el diagrama**

Un **diagrama de Chapín** simple podría ser, en donde se pide al usuario que ingrese un valor, se realiza un cálculo y se muestra el resultado en pantalla.  
El siguiente gráfico representa el **diagrama de Chapín** dicho:

|  |
| --- |
| Mensaje "Ingrese el diámetro" |
| Cálculo del perímetro |
| Informe del perímetro |

Un [**programador**](https://www.glosarioit.com/Programador) más experimentado, sin embargo, puede rehacer el diagrama de una manera diferente, adaptándolo así a su nivel o a las necesidades que tenga para la planificación de un programa.

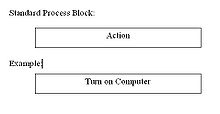
## **Un diagrama más rico**

Como se puede ver en este nuevo diagrama ya se incluyeron elementos más típicos de un programa, como son las variables **d** y **P**. Además, la fórmula que realiza el cálculo se ha incluido en el segundo paso del diagrama, de modo de no tener que pensarla luego, cuando llegue la hora de escribir el programa.  
Obviamente, esto supone que el [**programador**](https://www.glosarioit.com/Programador) cuenta con ciertos conocimientos y una determinada facilidad para interpretar el diagrama a la hora de codificarlo.  
A continuación el **diagrama de Chapín**:

|  |
| --- |
| "Ingrese el diámetro" **d** |
| **P** = π \* **d** |
| Informe **P** |

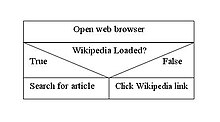
## **Tipos de diagramas**

**Bloques de procesos** El bloque de proceso representa el paso más simple y no requiere ningún análisis específico. Cuando un bloque de proceso es encontrado, la acción dentro del bloque se realiza, y pasamos directamente al siguiente bloque.

[](https://es.wikipedia.org/wiki/Archivo:Process_Block2.jpg)

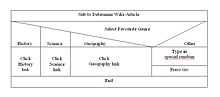
Process blocks

**Bloques ramificados** hay dos tipos de estos bloques. El primero y más sencillo de ellos es el bloque **verdadero-falso** el cual ofrece al programa dos caminos para tomar, dependiendo de si una determinada condición ha sido especificada. Estos bloques pueden ser usados como bucles que detienen el programa hasta que una determinada condición se cumpla.

[](https://es.wikipedia.org/wiki/Archivo:Two-Way_Branch.jpg)

True/False branching blocks

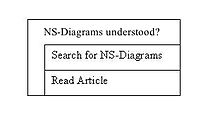
El segundo tipo es un bloque ramificado múltiple. Este tipo de bloque es utilizado cuando se necesita la selección de un caso en un programa. El bloque suele contener una pregunta. Además, el bloque le da al programa una cadena de oportunidades y es generalmente usado en las conjunciones con bloques de subprocesos para ahorrar espacio.

[](https://es.wikipedia.org/wiki/Archivo:Multiple_Branching.jpg)

Múltiple branching blocks

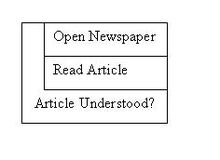
**Bucles testadores:** este bloque permite al programa repetir un bloque o un conjunto de bloques hasta que una determinada condición se haya cumplido.

Hay dos tipos de estos bloques: de testeo inicial y testeo final. La única diferencia entre los dos es el orden en el cual se completan los pasos involucrados en el proceso. En los de la primera situación, cuando el programa encuentra el bloque, testea si la condición necesaria se cumple, si no, se repite el bucle. El test se repite hasta que se cumpla dicha condición. En el nivel que se cumpla la condición, el programa detiene la ejecución del bucle y pasa a analizar los bloques del siguiente nivel.

[](https://es.wikipedia.org/wiki/Archivo:Test_First.jpg)

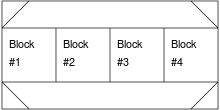
Test first loop block

Los de testeo final operan al revés.

[](https://es.wikipedia.org/wiki/Archivo:Test_Last.jpg)

Test lats loop block

Una **expresión concurrente** puede ser dibujada así:[5](https://es.wikipedia.org/wiki/Diagrama_Nassi-Shneiderman#cite_note-5)​

[](https://es.wikipedia.org/wiki/Archivo:NassiShneiderman-Parallel.svg)

parallel processing