Clase 3

Agenda

01 Tipos de Variables

02 La Memoria

03 Estructuras de Datos

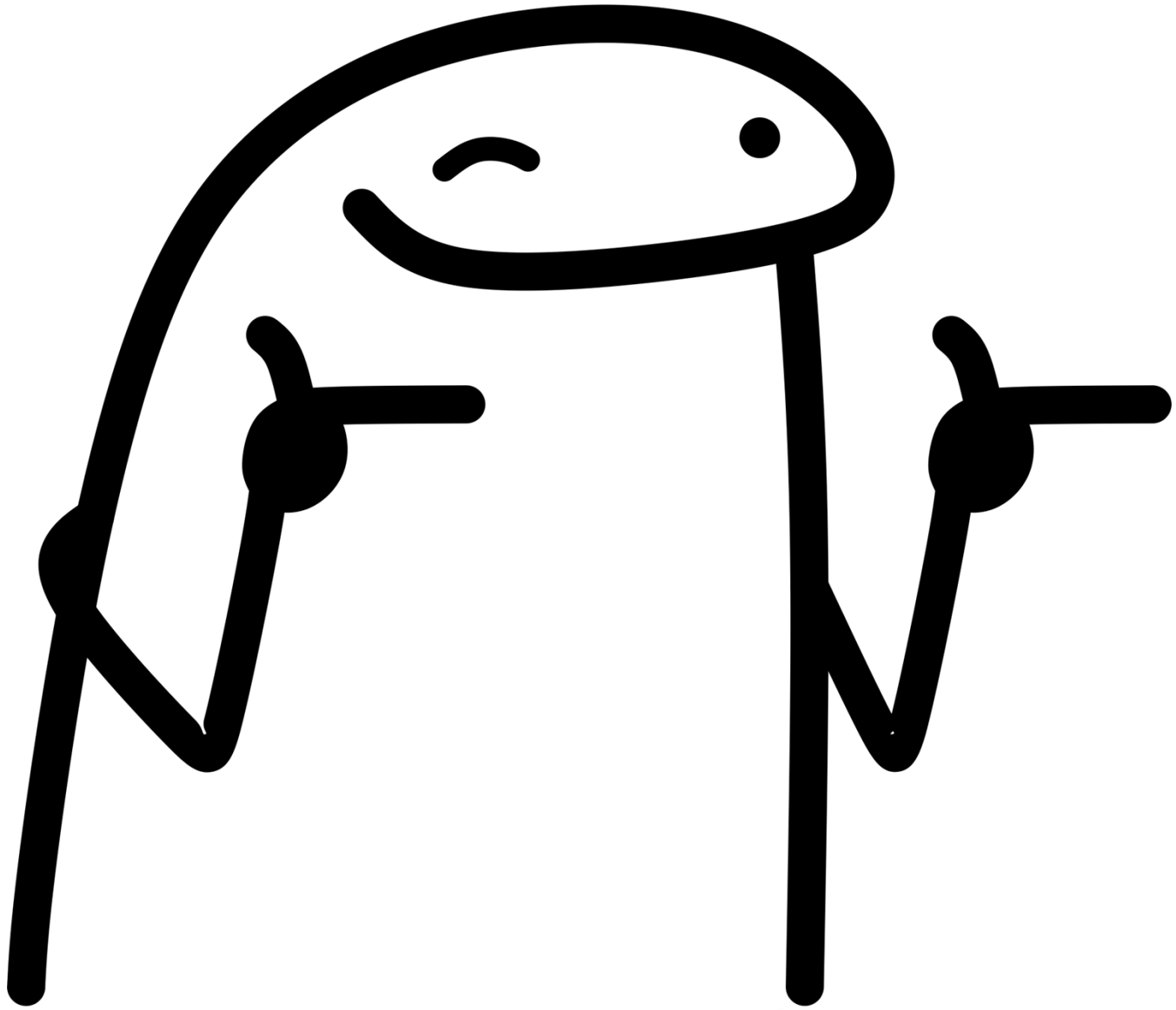
04 Arreglos

05 Listas

06 Archivos Planos

Tipos de variables

Ámbitos, Variables en JAVA, Valor y Referencia

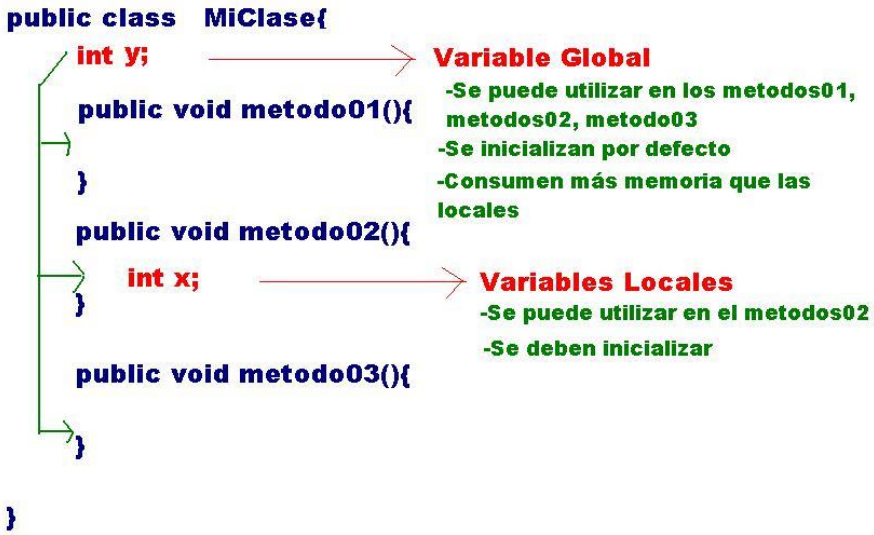


Ámbito en Programación

En programación de computadoras, el ámbito (referido mediante su denominación en inglés “scope”) es el contexto que pertenece a un nombre dentro de un programa. El ámbito determina en qué partes del programa una entidad puede ser usada. Esto sirve para que se pueda volver a definir una variable con un mismo nombre en diferentes partes del programa sin que haya conflictos entre ellos.

Variables Globales

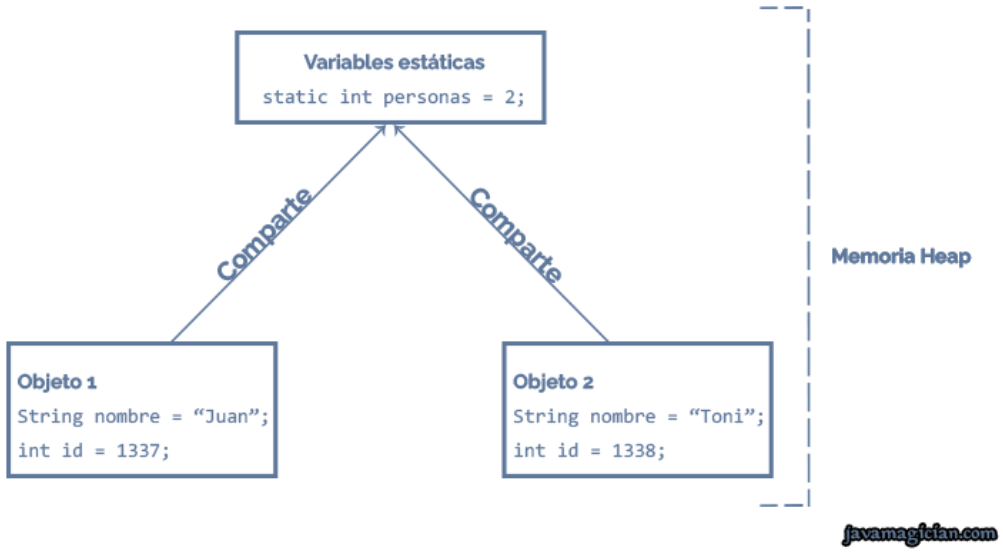
Una variable global es una variable accesible en todos los ámbitos de un programa informático. Los mecanismos de interacción con variables globales se denominan mecanismos de entorno global. El concepto de entorno global contrasta con el de entorno local donde todas las variables son locales sin memoria compartida (y por ello todas las iteraciones pueden restringirse al intercambio de mensajes).



Variables Estáticas

De Instancia

Una variable estática es una variable que ha sido ubicada estáticamente y cuyo tiempo de vida se extiende durante toda la ejecución del programa. Normalmente una variable estática tiene un ámbito más amplio que otras variables. Los valores de variables estáticas se pueden establecer una vez (durante el tiempo de ejecución) o se pueden cambiar en múltiples ocasiones durante la ejecución del programa. La terminología "variable estática" se basa en C y C++, pero también se usa en muchos lenguajes de programación derivados.



Variables Finales

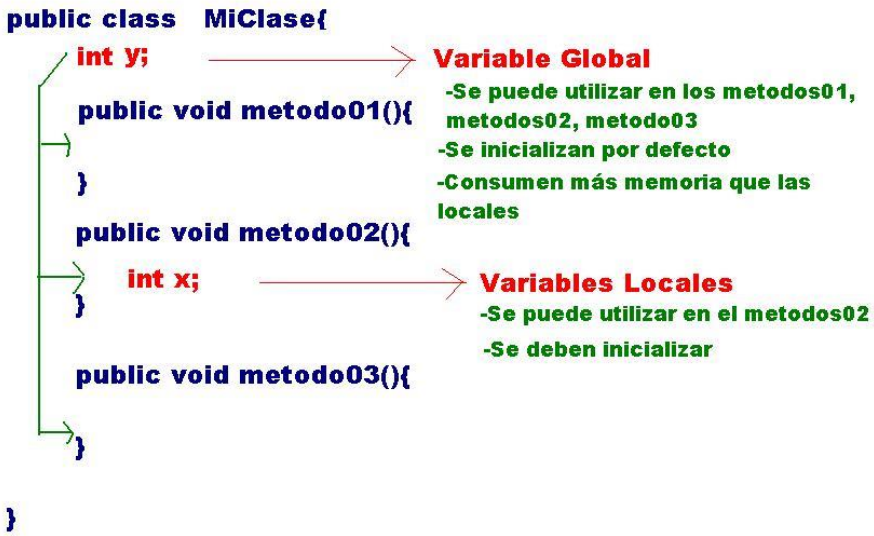
Una variable final se utiliza para definir una entidad que puede asignarse una única vez, y permanecerá siempre con el mismo valor. Si una variable final referencia a un objeto, los atributos de la instancia del objeto pueden cambiar, pero la variable final siempre hará referencia a la misma instancia.

```
class FinalClass
{
    final int Num = 5;

    class InheritedClass extends FinalClass
    {
        {
ss.Num cannot be assigned Num = 6;
        }
    }
};
```

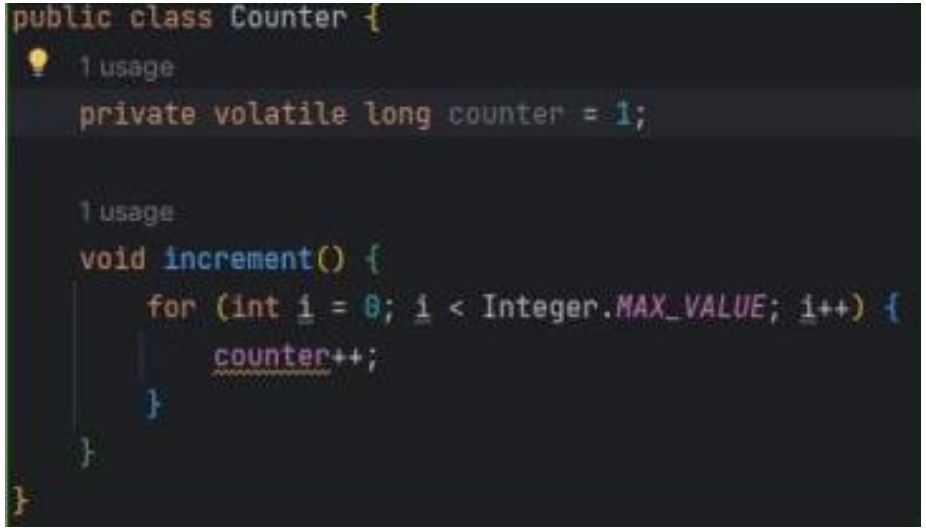
Variables Locales

Una variable local es la variable a la que se le otorga un ámbito local. Tales variables sólo pueden accederse desde la función o bloque de instrucciones en donde se declaran. Las variables locales se contraponen a las variables globales.



Variables Volátiles

Son variables que se marcan en Java para ser almacenadas en la memoria principal. Significa que cada lectura a una variable volátil será desde la memoria principal de la computadora, y no del caché del CPU, y toda escritura de una variable sucederá en la memoria principal de la computadora, y no en el caché del CPU. Garantiza que las variables volátiles sean escritas y leídas desde la memoria principal.



Variables Trascendentes

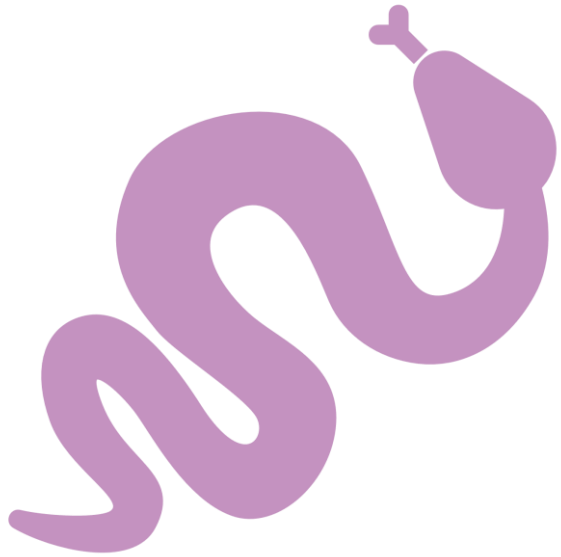
Son variables se utilizan para evitar la serialización. Si una variable está definida como trascendente, esta no será serializada. Se llama "serializar un objeto" al proceso de convertirlo a bytes, para poder enviarlo por una red, y reconstruirlo luego a partir de esos bytes.

Tipos Enumerativos

Tipo de dato que consiste en una serie de valores con nombres propios, llamados numerables. Los numerables suelen tener valores constantes, y se utilizan para tareas de comparación propia. Son declarados para optimizar tareas de identificación.

Valor y Referencia

En Java, una variable adquiere su valor por medio de una asignación. Depende del tipo de la variable, el valor que adquiere la misma como una copia o una referencia hacia el valor obtenido. Los tipos de datos primitivos y las cadenas en Java se manejan por valor, mientras que los objetos se manejan por referencia.



Ejemplo de Valor y Referencia

Asignación por Valor

- int a = 5;
  int b = a;
  print(b);

Resultado:

>> 5

Asignación por Referencia

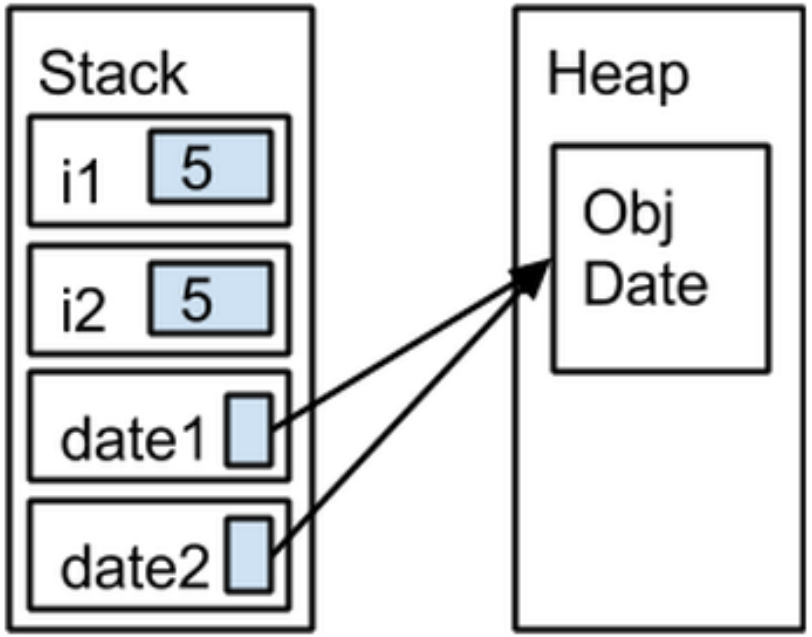
- int[] a = {1, 2, 3};
  int[] b = a;
  a[0] = 100;
  print(b);

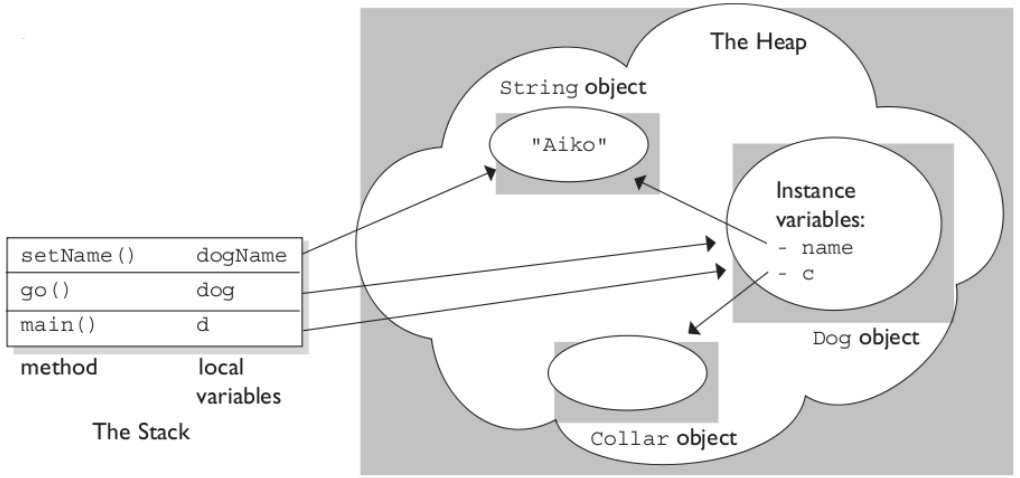
Resultado:

>> 100, 2, 3

La Memoria

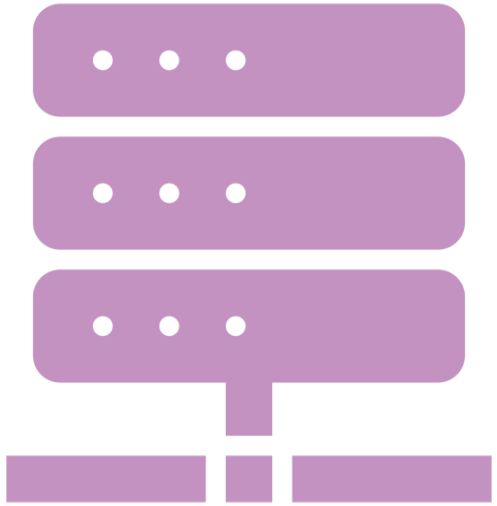
Definición





Memoria

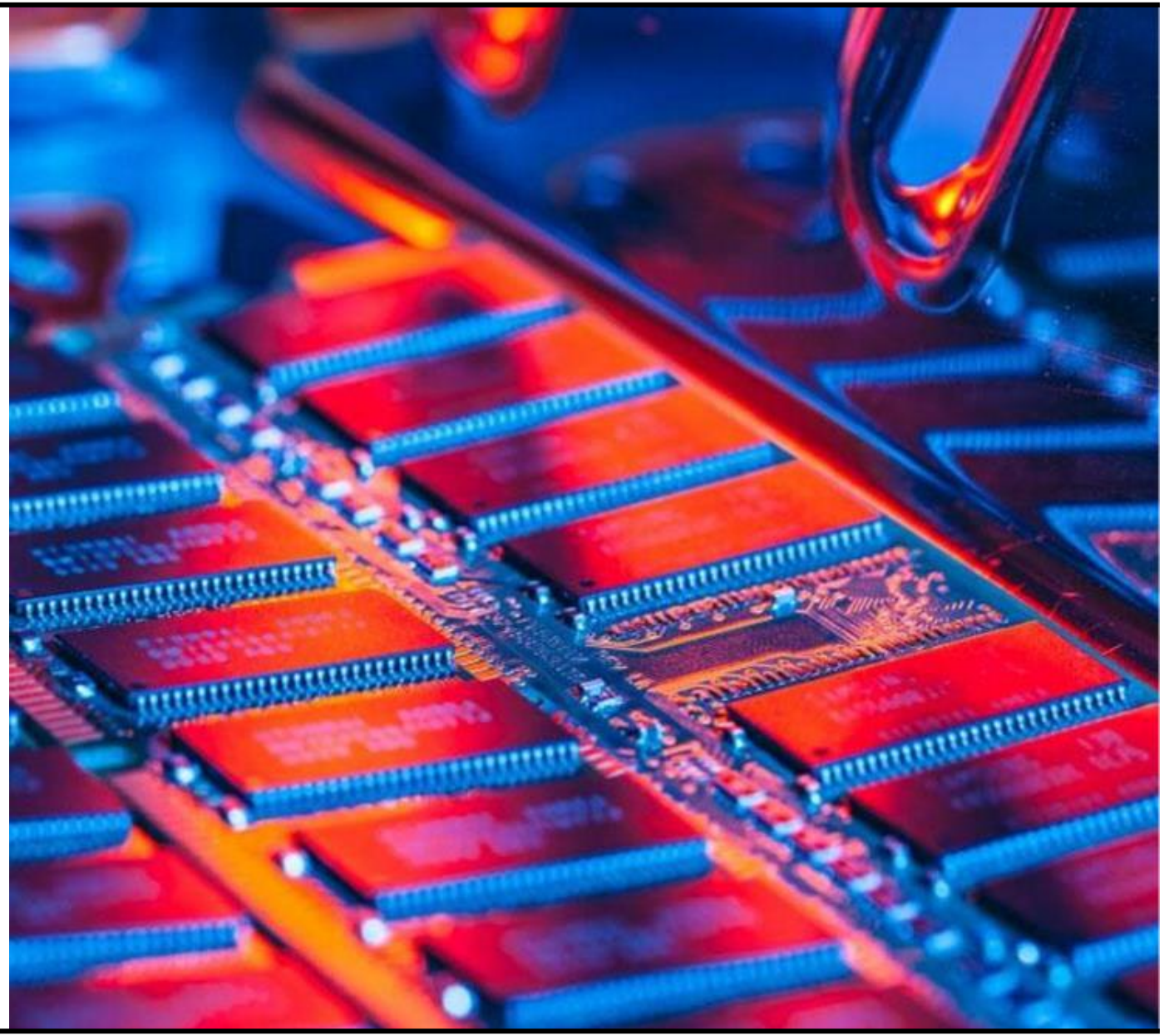
Es el dispositivo que retiene, memoriza o almacena datos informáticos durante algún periodo de tiempo. La memoria proporciona una de las principales funciones de la computación moderna: el almacenamiento de información y conocimiento. Es uno de los componentes fundamentales de la computadora, que interconectada a la unidad central de procesamiento y los dispositivos de entrada/salida, implementan lo fundamental del modelo de computadora de la arquitectura de von Neumann.



Memoria

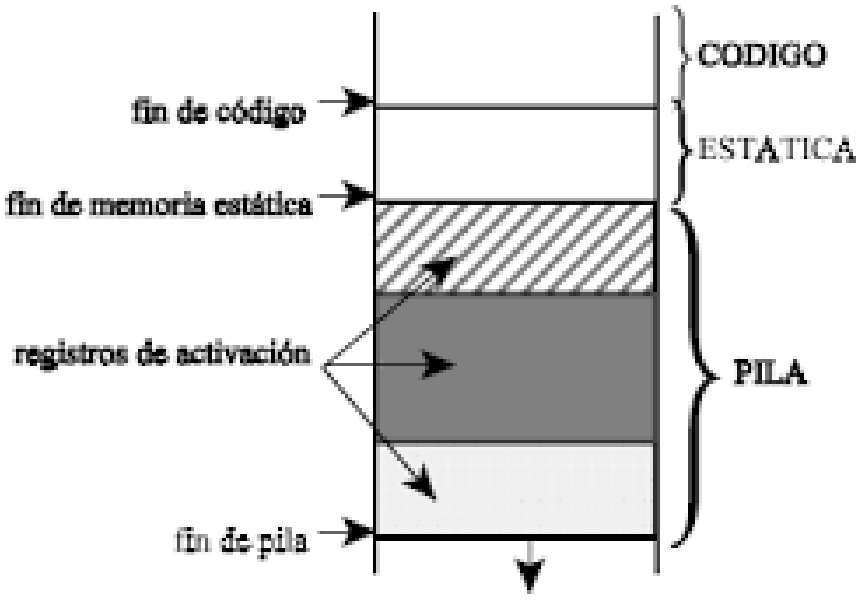
La ejecución de un programa requiere que diversos elementos se almacenen en la memoria:

- Código del programa (Instrucciones)
- Datos
  - Permanentes
  - Temporales
- Direcciones para controlar de flujo de ejecución del programa



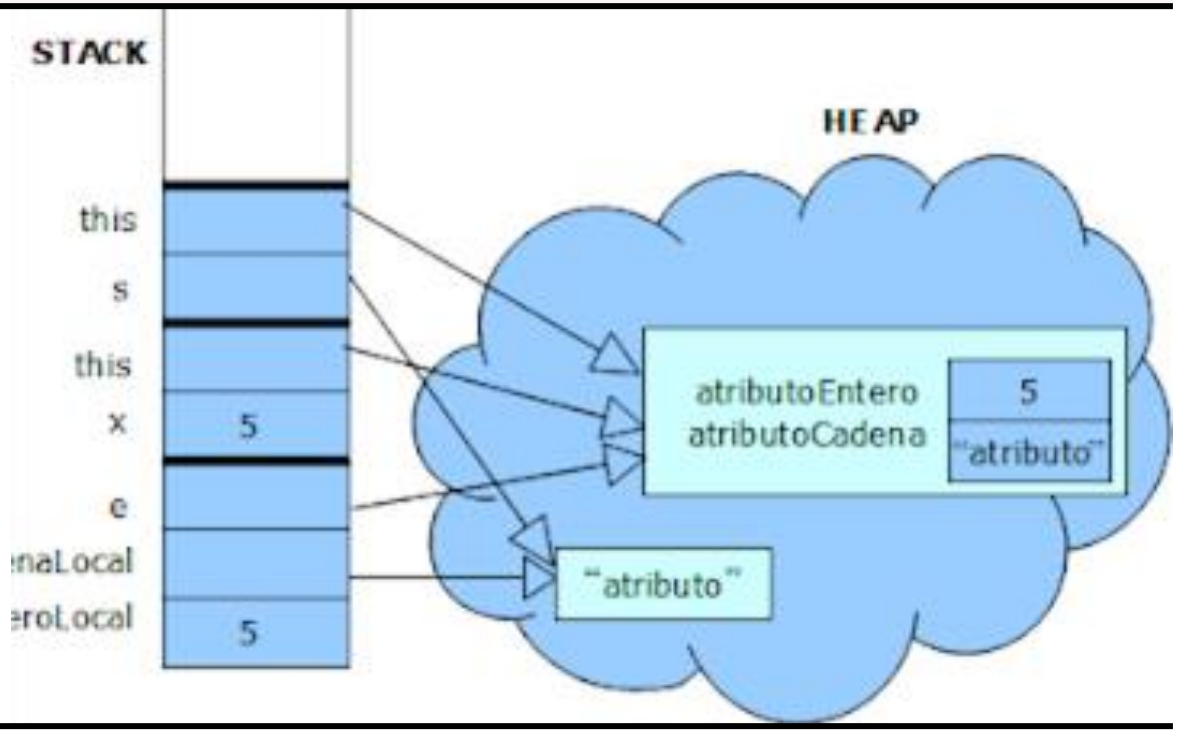
Memoria Estática

Es aquella que no se modifica en tiempo de ejecución, es decir, La asignación de memoria puede hacerse en tiempo de compilación y los objetos están vigentes desde que comienza la ejecución del programa hasta que termina.



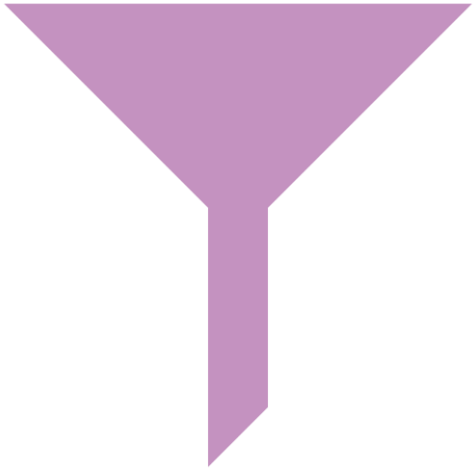
Memoria Dinámica

Es aquella que se modifica permanentemente, es decir, a medida que el proceso va necesitando espacio para más líneas, va solicitando más memoria al sistema operativo para guardarlas.

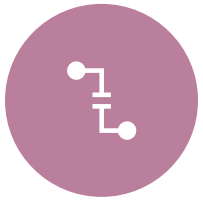


Stack

Cuando un programa es ejecutado, a cada hilo se le asigna una cantidad limitada de espacio en el Stack, el cuál se encarga de almacenar la información usada por el programa incluyendo el byte-code ejecutado por el procesador. Se trata de un fragmento de memoria donde se van apilando linealmente (estructura LIFO) las distintas funciones ejecutadas así como las variables locales de cada una de ellas. El modo en como se asignara la memoria en el Stack, se define durante el proceso de compilado.

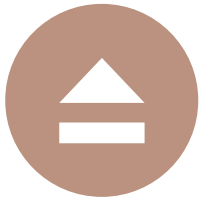


Consideraciones del Stack



Las variables almacenadas en el Stack (o variables automáticas) son almacenadas directamente a esta memoria.

Su acceso es muy rápido.



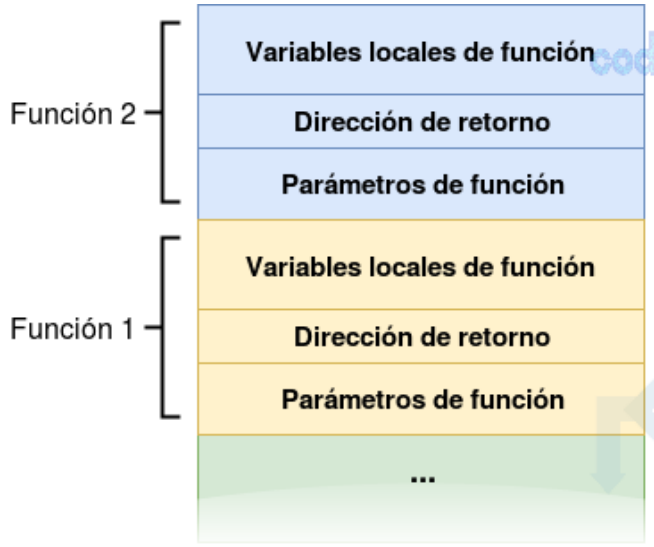
Es liberada al terminar la ejecución.

Fragmentos grandes de memoria, como arrays de gran envergadura, no deberían ser almacenados en el Stack, para prevenir desbordamientos del mismo (Stack Overflow).

Las variables almacenadas en el Stack solamente son accesibles desde el bloque de código donde fueron declaradas.

Ejemplo Stack

```
void main(){
    if (true) {
          int x = 0;
          }
    x = 1;
}
```

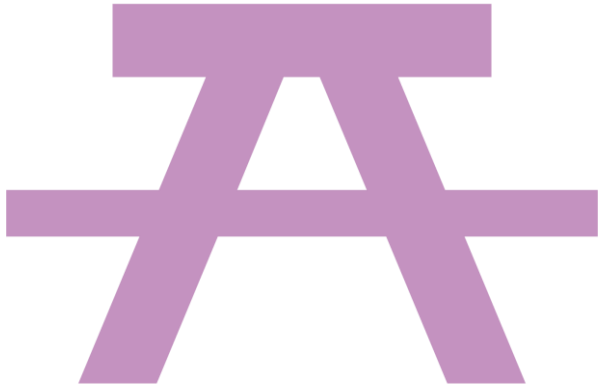


Ejemplo de pila de llamadas (*call stack*)

Supongamos que: "Función 0" ha llamado a "Función 1" y esta a su vez ha llamado a "Función 2"

Heap

El heap (o montículo libre) es una estructura dinámica de datos usada para almacenar información en tiempo de ejecución. A diferencia de la pila, que almacena variables locales, el heap permite la reserva y liberación dinámica de memoria. También alberga variables globales y estáticas. Sin embargo, su manejo es más complejo, ya que los bloques de memoria pueden ser ocupados y liberados en cualquier momento sin seguir un orden específico.

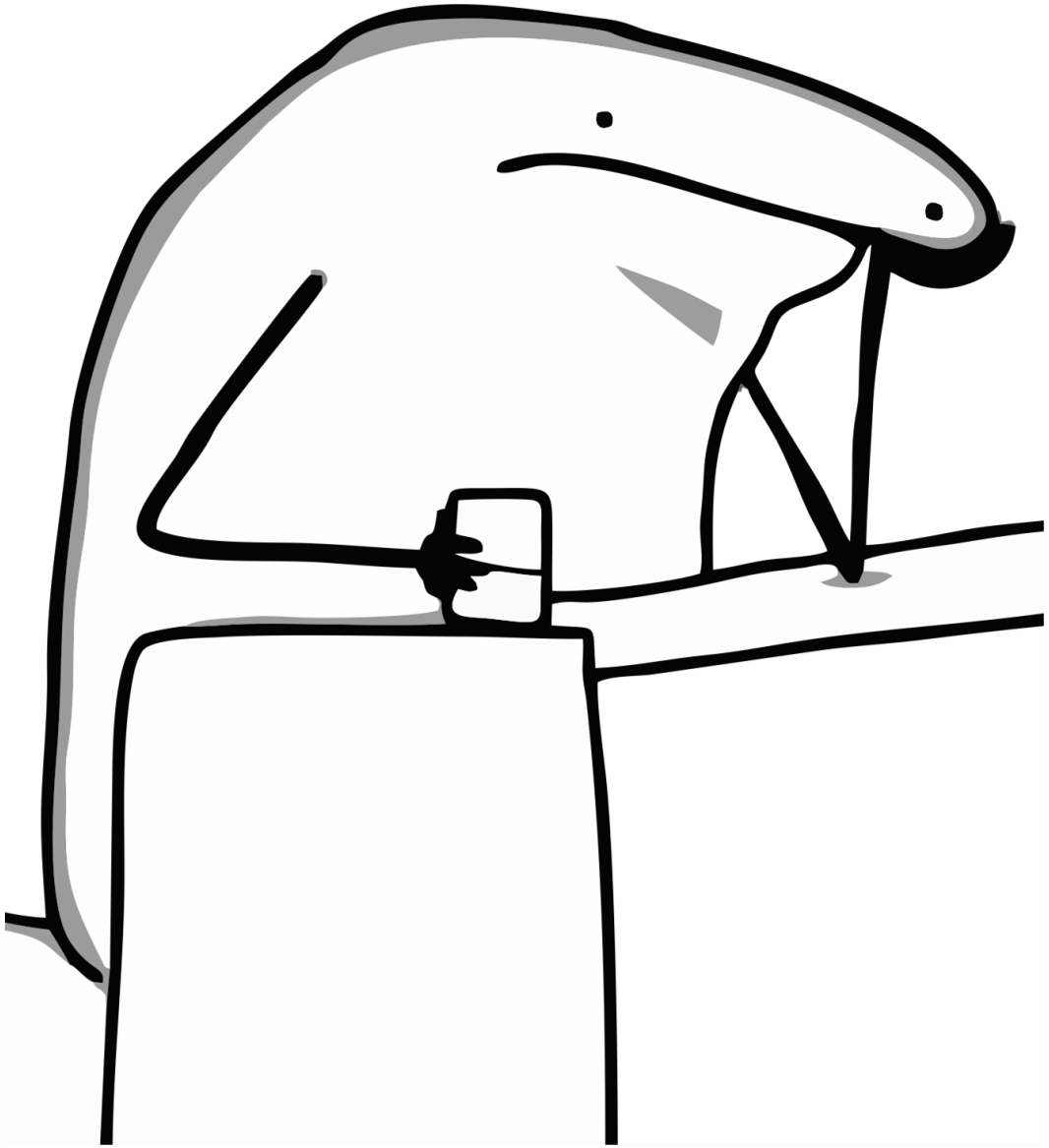


Stack Vs Heap

Aspecto	Stack (Pila)	Heap (Espacio Libre)
Organización	Estructura ordenada tipo LIFO (Last In, First Out).	Estructura desordenada; bloques se manejan dinámicamente.
Almacenamiento	Variables locales, parámetros de funciones, y datos temporales.	Objetos, variables globales, estáticas, y memoria dinámica.
Reserva de Memoria	Automática durante la ejecución de funciones.	Manual mediante operaciones como malloc o new.
Tamaño	Limitado y definido en tiempo de compilación.	Mayor y definido en tiempo de ejecución.
Velocidad	Más rápido debido a su organización simple.	Más lento por la necesidad de manejar fragmentación.
Acceso	Directo y rápido.	Más complejo, requiere punteros para referencia.
Manejo	Administrado automáticamente por el sistema.	Requiere gestión explícita para asignar y liberar memoria.
Errores Comunes	Desbordamiento de pila (stack overflow).	Fugas de memoria (memory leaks).

Estructuras de Datos

¿Cómo guardar la información?



Definiciones

Estructura

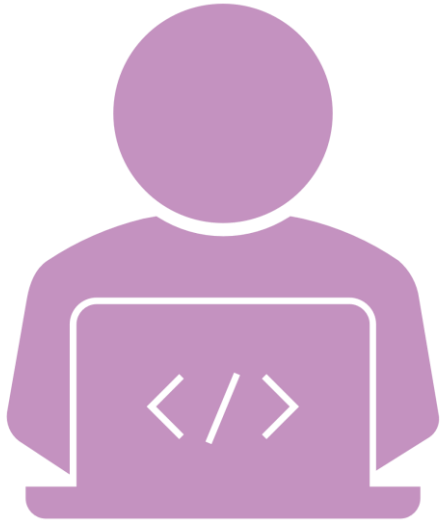
La estructura es el conjunto de elementos que caracterizan un determinado ámbito de la realidad o sistema. Los elementos estructurales son permanentes y básicos, no son sujetos a consideraciones circunstanciales ni coyunturales, sino que son la esencia y la razón de ser del mismo sistema.

Datos

Un dato es una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades. Es un valor o referente que recibe el computador por diferentes medios. Los datos representan la información que el programador manipula en la construcción de una solución o en el desarrollo de un algoritmo.

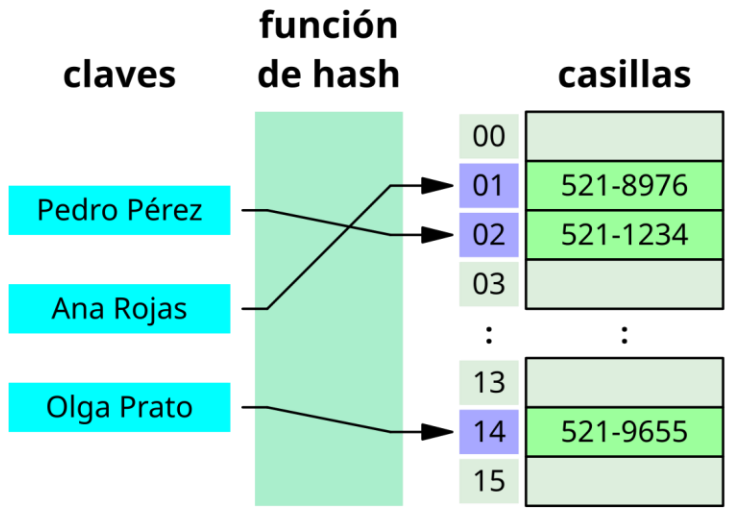
Estructura de Datos

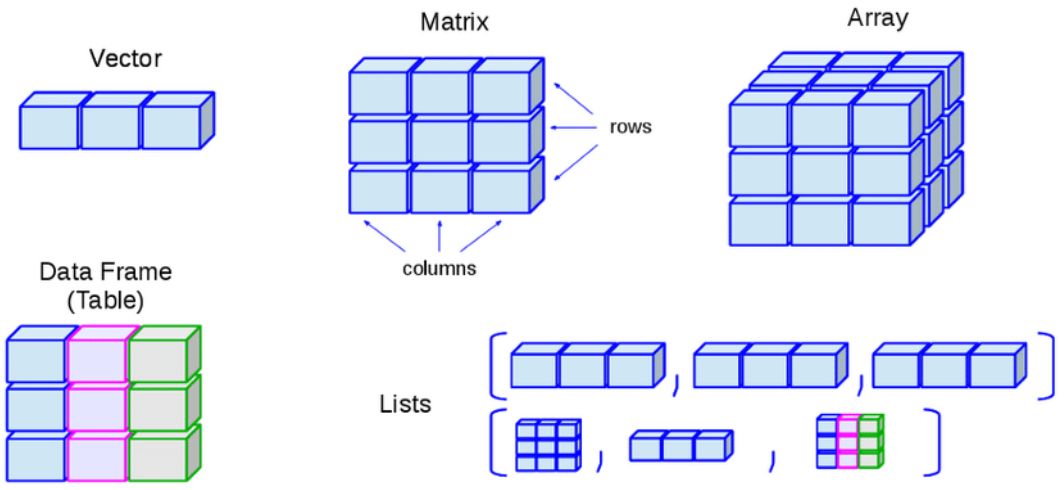
En ciencias de la computación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.



Ejemplos de Estructuras de Datos

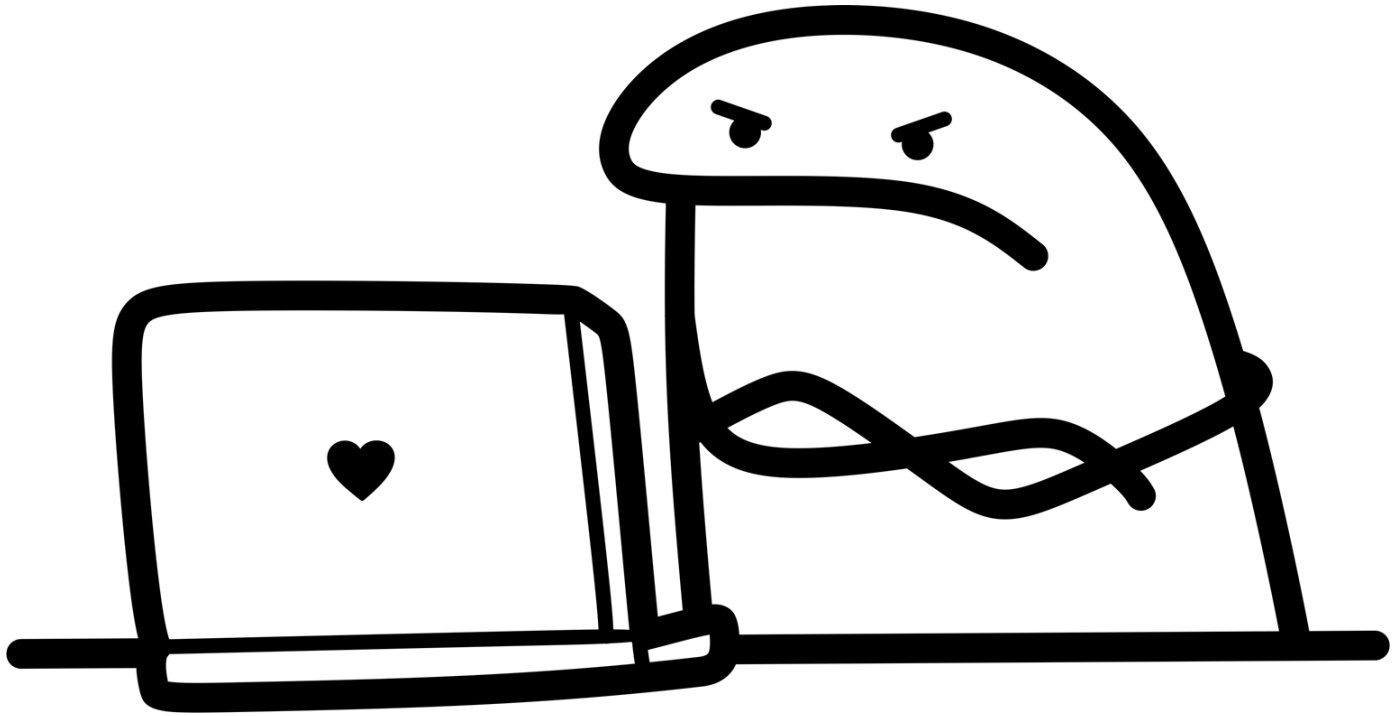
- Vector
- Arreglo
- Case
- Árbol
- Estructura (struct)
- Lista
- Cadena





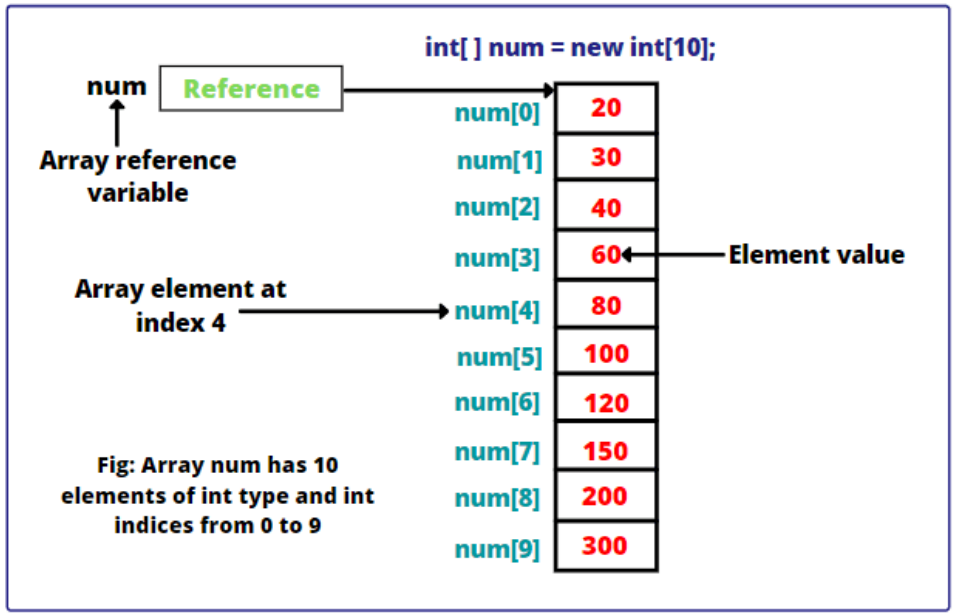
Arreglos

- Definición
- Sintaxis
- Valores Default
- Tips
- Métodos



Arreglos

Un array (arreglo) en Java es una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar luego durante la ejecución del programa, como sí puede hacerse en otros lenguajes.



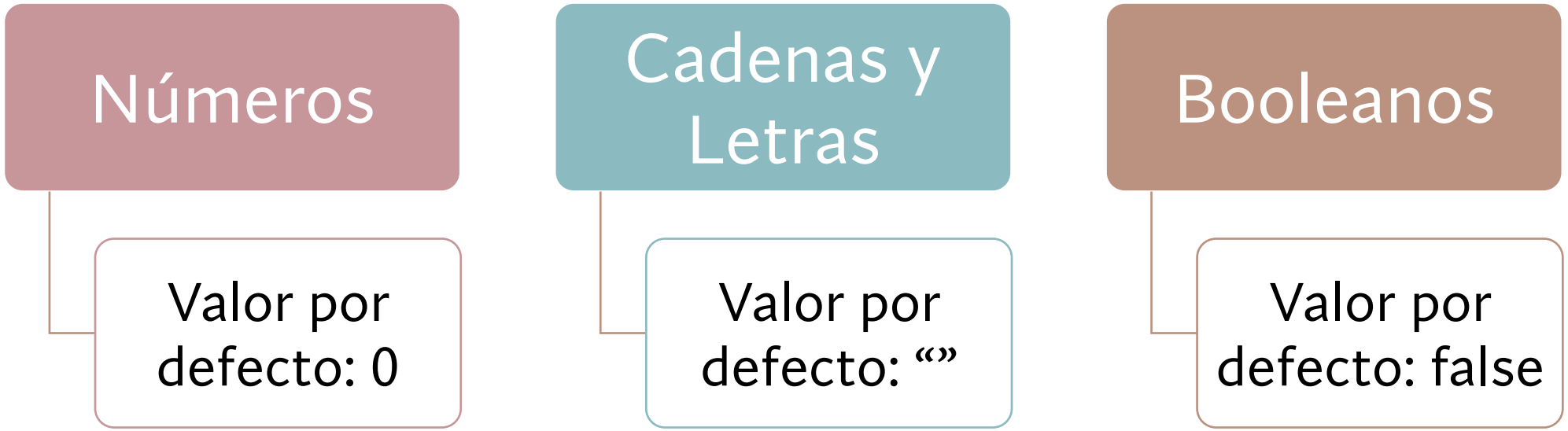
Sintaxis

<Tipo de dato> ”#[]” <ID> = new <Tipo de dato> #”[“<Longitud>”]”

Ejemplos

```
int[] numeros = new int[5];
int[][] matriz = new int[3][3];
string[] nombres = new string[12];
```

Valores por defecto en arreglos
(tipo de dato)



Tips para conocer Arreglos

Indices

- El índice comienza desde n = 0 y termina con el tamaño n – 1.

Verlo (Imprimirlo)

- Utilizar ciclo for para recorrer el arreglo e imprimirlo.

Acceso

- Nombre_arreglo[n]

Métodos para Arreglos

split

Divide una cadena en subcadenas utilizando un delimitador definido mediante una expresión regular o caracter.

toCharArray

Convierte una cadena en un array de caracteres.

Split vs toCharArray

split

String saludo = “hola mundo”;
String[] dividir = saludo.split(‘ ‘);

> [“hola”, “mundo”]

toCharArray

String saludo = “hola”;
Char[] dividir = saludo.toCharArray();

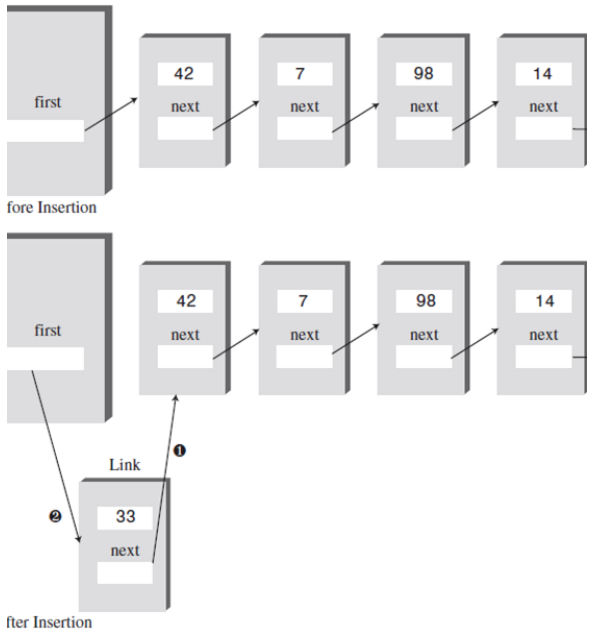
> [“h”, “o”, “l”, “a”]

Listas

Definición, ordenamientos

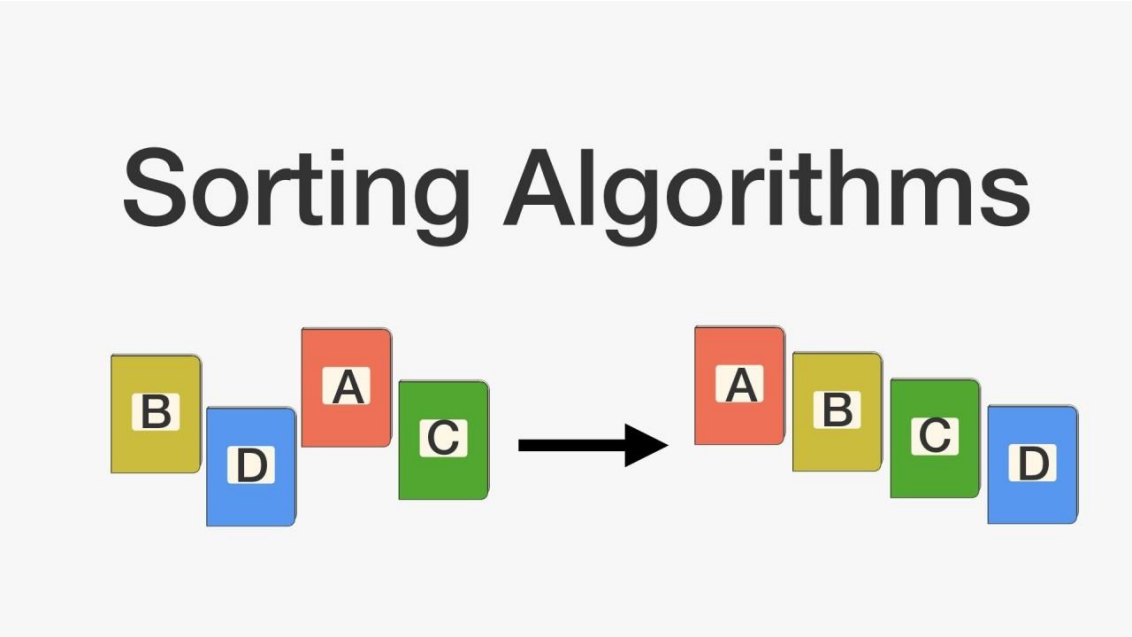
Listas

La lista tiene un enlace al primer contenedor y cada contenedor tiene un enlace al siguiente contenedor en la lista. Para agregar un elemento a la lista, el elemento se coloca en un nuevo contenedor y ese contenedor está vinculado a uno de los otros contenedores en la lista.



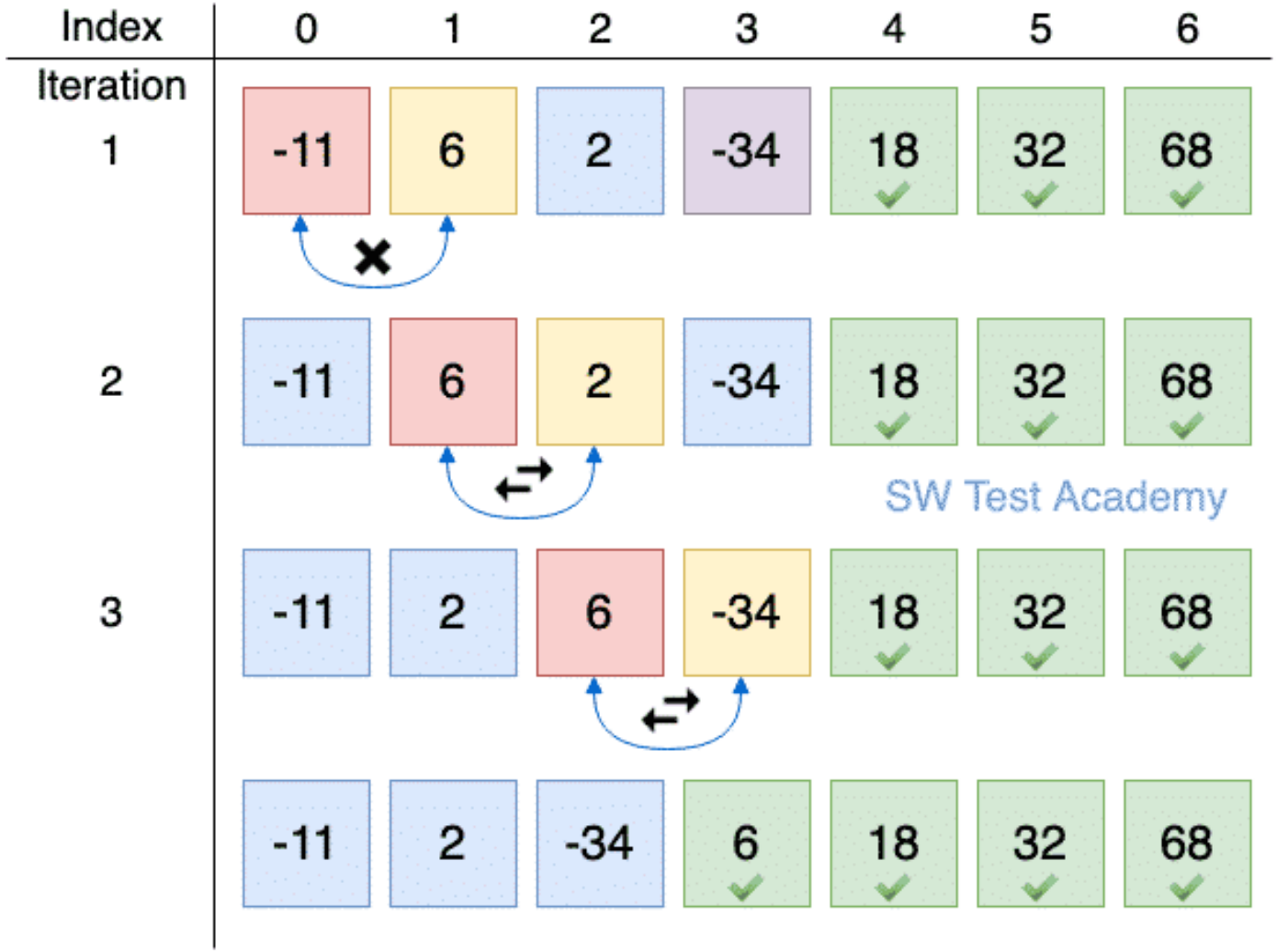
Ordenamiento

Es la operación de arreglar los elementos en algún orden secuencial de acuerdo a un criterio de ordenamiento. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado. Ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia por categorías y en forma ascendente o descendente.



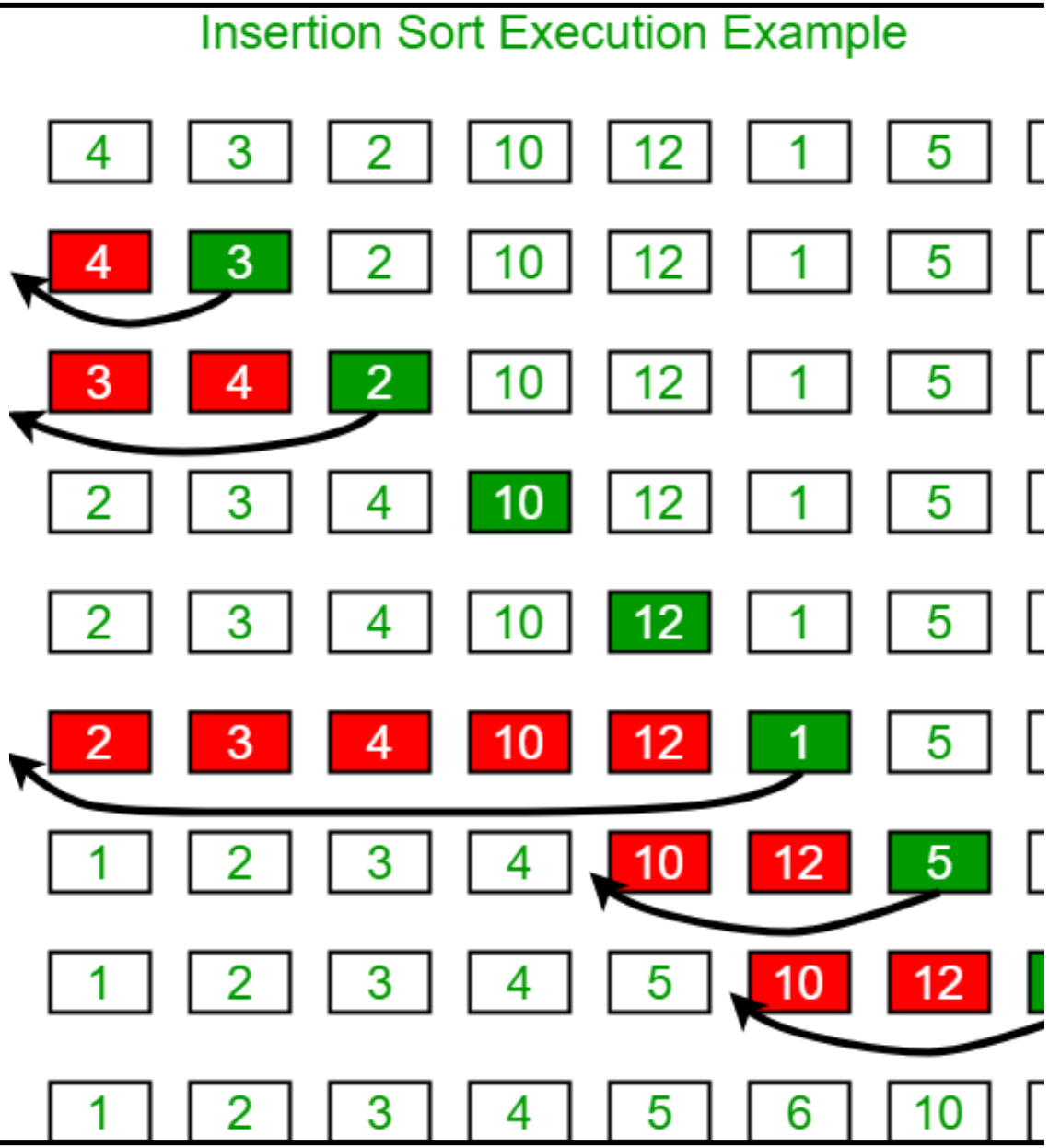
Ordenamiento Burbuja

El ordenamiento burbuja hace múltiples pasadas a lo largo de una lista. Compara los ítems adyacentes e intercambia los que no están en orden. Cada pasada a lo largo de la lista ubica el siguiente valor más grande en su lugar apropiado. En esencia, cada ítem “burbujea ” hasta el lugar al que pertenece.



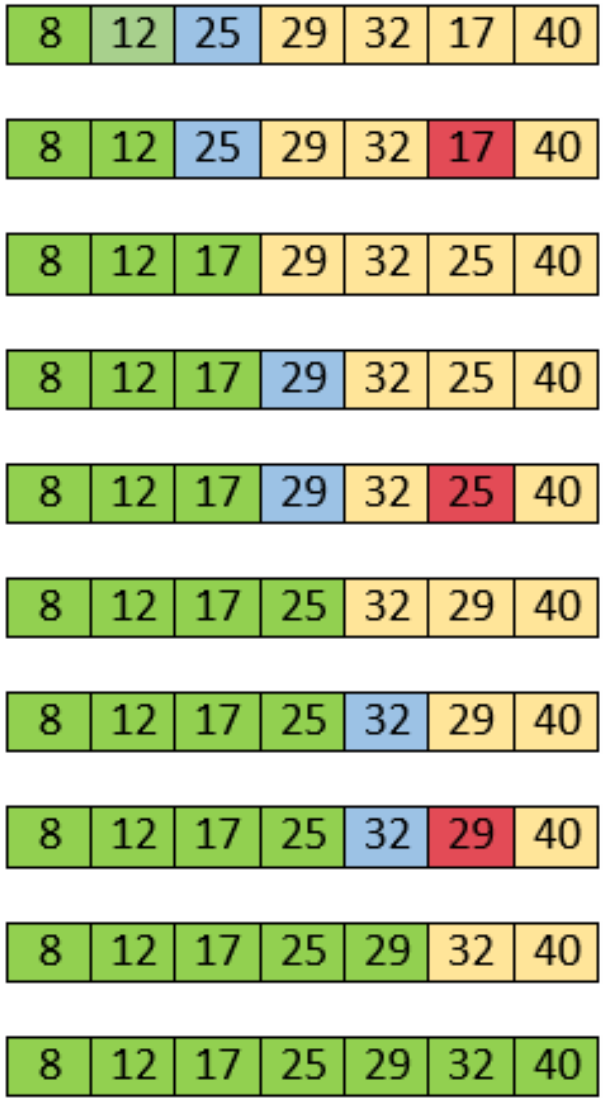
Ordenamiento por Inserción

En este tipo de algoritmo los elementos que van a ser ordenados son considerados uno a la vez. Cada elemento es “insertado ” en la posición apropiada con respecto al resto de los elementos ya ordenados.



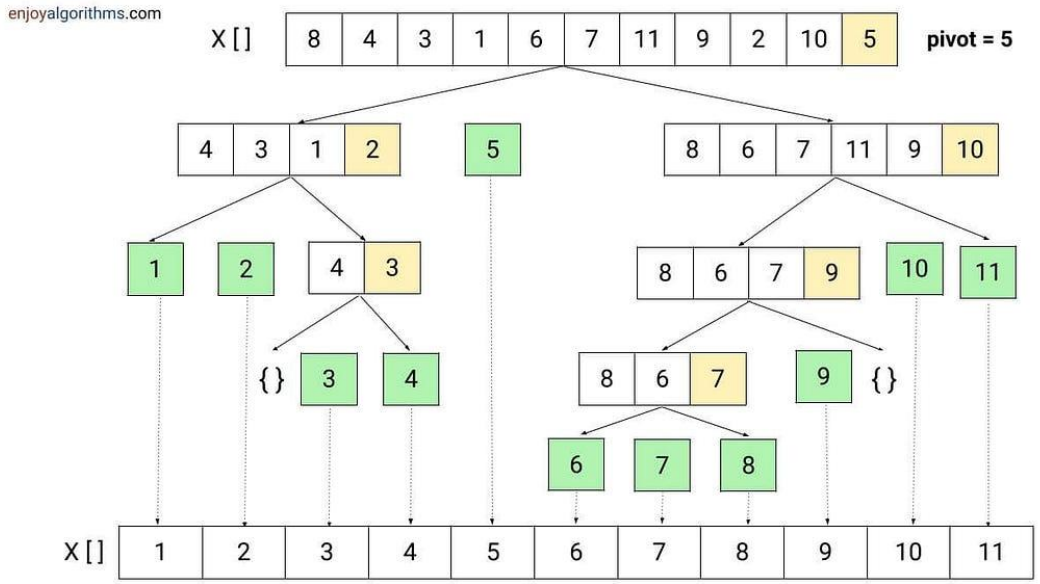
Ordenamiento por Selección

El ordenamiento por selección mejora el ordenamiento burbuja haciendo un sólo intercambio por cada pasada a través de la lista. Para hacer esto, un ordenamiento por selección busca el valor mayor a medida que hace una pasada y, después de completar la pasada, lo pone en la ubicación correcta. Al igual que con un ordenamiento burbuja, después de la primera pasada, el ítem mayor está en la ubicación correcta. Después de la segunda pasada, el siguiente mayor está en su ubicación.



Ordenamiento Rápido

El método Quick Sort es actualmente el mas eficiente y veloz de los método de ordenación interna. Es tambien conocido con el nombre del método rápido y de ordenamento por partición. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de Quick Sort, por la velocidad con la que ordena los elementos del arreglo. Quicksort es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n.



Dudas o Comentarios