# 1.Fundamentos De Programación: Lógica y Algoritmos

**Nota:** Los apuntes aquí mostrados no tienen como objetivo utilizar o darle visibilidad a un lenguaje de programación en específico sino a conocer la lógica que luego podremos extrapolar a casi cualquier lenguaje

Además, apuntes sobre buenas prácticas a la hora de diseñar/escribir programas que nos ayudan a hacer software de mejor calidad, reduciendo tiempo y costos a la hora de la operación de estos programas

## 1.2Lista De Definiciones y Algunos Conceptos

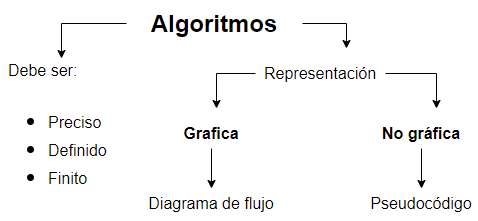
* **Palabras reservadas**: Una palabra reservada es una palabra que tiene un significado gramatical especial para ese lenguaje y no puede ser utilizada como un identificador de objetos en códigos del mismo, como pueden ser las variables

## 1.3. Path De Aprendizaje De Programación Básica Para Niños:

Estas son las bases que pienso que de manera general deberían enseñárseles a los niños de manera conjunta para que aprendan programación:

1. Mecanografía
2. Bases matemáticas (Aritmética)
3. Algoritmos básicos: Diagramas de flujo code.org
4. HTML
5. PHP
6. CSS

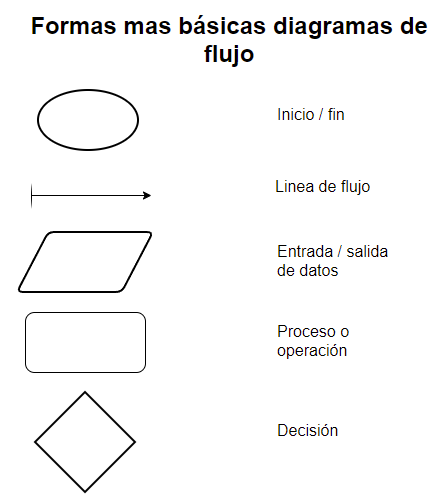
# 2.Algoritmos



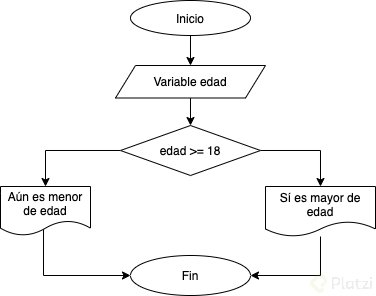
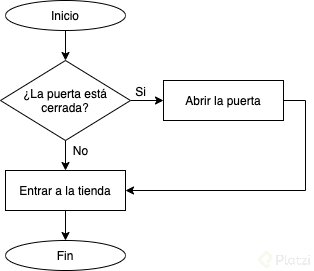
Para elaborarlos debemos

* Definir el problema ¿Cuál es su objetivo principal?
* Analizar el problema
* Diseñar el algoritmo
* **Prueba de Escritorio**: Probar nuestro manualmente y contrastar estos resultandos con los que nos devolvería el algoritmo en la máquina. En scheme una herramienta para hacer pruebas de escritorio es (chechk-expect)

## 2.1 ¿Cómo diseñar un algoritmo? – Diagramas de flujo



Ejemplos de diagramas de flujo

## 2.2 Variables y Constantes

* **Variables**: Es un espacio en la memoria de nuestro pc para almacenar un dato
* **Constante**: Es también un espacio en memoria, pero una vez lo creamos (lo llenamos con un dato) no lo podemos volver a cambiar

Las variables pueden almacenar cualquier tipo de dato que soporte un lenguaje de programación

### 2.2.1 Acumulador

En programación un acumulador es una variable donde almacenamos operaciones temporales que hacemos, los ejemplos más comunes de acumulador son cuando utilizamos ciclos o recursión donde el acumulador funciona como un dato temporal que debemos modificar en cada “vuelta”. Esto se hace para evitar el fenómeno llamado como perdida de conocimiento

### 2.2.2 Scope de una variable (Ámbito o alcance léxico)

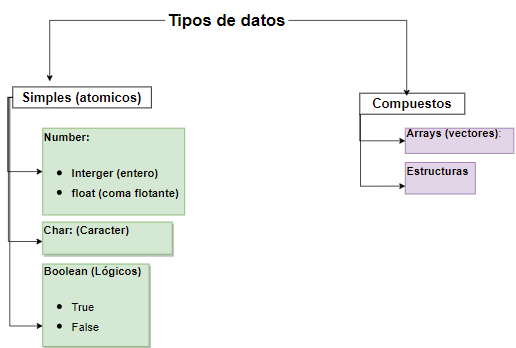
Es el contexto al que perteneciente un nombre (Variable dentro del programa) es decir el ámbito lo que determina es en que partes de nuestro programa podemos usar una variable

*Por ejemplo*: Si una variable es declarada dentro de un bloque (método/función/procedimiento), esta será válida solo dentro de ese bloque y se destruirá al terminar el bloque. Esto en términos generales, aunque cada entorno y lenguaje de programación tiene sus particularidades

# 

# 3.Tipos de datos

# Es un atributo de los datos que el programador le indica a la maquina sobre la clase de datos que se va a manejar, esto implica restricciones para los tipos de datos, pues todos tienen un objetivo distinto



### 3.1 Tipos de datos simples

También llamados atómicos, primitivos, escalares o elementales, son tipos de datos donde la unidad mi mínima de almacenamiento es el dato

#### 3.1.1 Numero (Number)

Son números con los que podemos hacer operaciones matemáticas

* **Enteros (Interger)**: Son números que no tienen parte decimal
* **De coma flotante (float, decimales)**: Tienen una parte entera y otra decimal

Las operaciones más básicas que les podemos aplicar son con los operadores aritméticos: + - \* /

#### 3.1.2 Carácter (Char)

Es una unidad de información que se compone por un carácter, este carácter puede referirse a caracteres alfanuméricos, símbolos, caracteres de control, emojis etc… la codificación mas popular de caracteres es Unicode

* Tanto el carácter como la cadena de caracteres se escriben en medio de “” ‘’ comillas dobles y/o comillas simples, esto ya depende de la sintaxis del lenguaje

1. Ej:
2. 

#### 3.1.3 Booleanos (Booleans o lógicos)

Estos valores nos permiten tomar decisiones y modificar el comportamiento de un programa mediante las estructuras de control, generalmente los obtenemos como resultando de una condición o comparación en una estructura de control, los dos valores posibles de un booleano son:

* True
* False

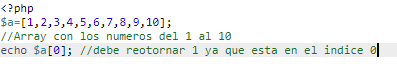
En algunos lenguajes como MySQL no existen este tipo de dato y en lugar de ello se recomienda usar el 1 como true y 0 como false

### 3.2 Tipos de datos compuestos

Son la representación del concepto matemático de conjunto en la programación, los tipos de datos compuestos son un conjunto en el cual podremos almacenar otros tipos de datos simples (vendrían siendo los elementos) y/o otros tipos de datos compuestos que (vendrían siendo subconjuntos)

#### 3.2.1 Arrays (vectores)

Conjuntos de elementos del mismo tipo ordenandos en fila, los arrays pueden contener otros arrays, cada elemento de un array tiene un índice es decir un numero o nombre (array asociativo) por medio del cual podremos acceder a un elemento del mismo array



Ej:

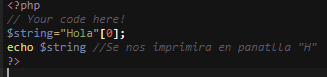
Podemos almacenar un array dentro de un array sucesivamente con lo cual formamos otra dimensión

\* Los arrays los podremos recorrer mediante ciclos

#### 3.2.2 Cadena de caracteres (String o character set)

Un string es en realidad un tipo de datos compuesto ya que se compone de varios caracteres, de hecho, muchos lenguajes de programación se refieren a la string como un arrray o lista de caracteres, donde podremos acceder a un carácter con su índice

Ej. (Lenguaje PHP):



#### 3.2.3 Estructuras:

Es un conjunto de elemento de distinto tipo

#### 3.2.3 Listas

La lista es una representación computacional del concepto matemático de una secuencia finita,

##### 3.2.3.1 Listas enlazadas

Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

En el caso de *racket/schem/Lisp* las listas están construidas con el operador (cons <elemento> <puntero al próximo elemento>)



En Racket esta lista se construiría así

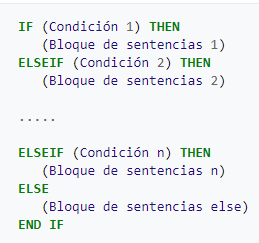
# 4.Estructuras de control

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa. La mayoría de las estructuras de control poseen una condición que al cumplirse ejecutará un bloque de código o dejara de ejecutar el mismo está ya depende de cada caso en específico, los elementos más comunes que componen estas condiciones son:

* Operadores de comparación: < > <= >= ==
* **Operadores lógicos**: and (conjunción), or (disyunción), xor (disyunción exclusiva) y not (negación)

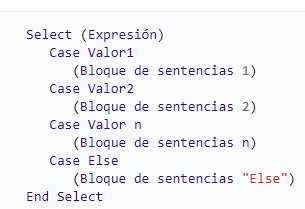
## 4.1 Condicionales

### 4.1.1 If/elseif/else



### 4.1.2 Switch Case

Nos permite evaluar una variable y en base a ella evaluarla con n-condiciones o n-caso para los que la variable toma el valor que asignemos en el condicional, de cumplirse alguna de estas condiciones se ejecuta el bloque de código asociado a ellas, si no se ejecuta un caso “default”



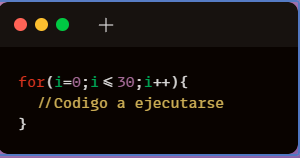
## 4.2 Ciclos (bucles o loops)

Es una estructura que nos ayuda a ejecutar un bloque de código varias varias veces según se cumpla una condición.

**Ciclo infinito**: Debemos tener cuidado en que siempre sea posible cumplir la condición para que sea posible salir de nuestro ciclo si no entraremos en un ciclo infinito que impedirá la ejecución de las siguientes instrucciones en nuestro programa

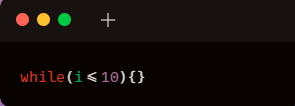
### 4.2.1 for

Nos permite ejecutar un bloque de código n-cantidad de veces:



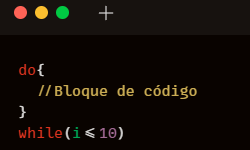
### 4.2.2 While (mientras)

Trabaja bajo una condición mientras esta se cumpla se ejecutará un bloque de código



### 4.2.3 Do-while

Este se ejecuta por lo menos una sola vez y luego verifica la condición y si esta se sigue cumpliendo vuelve a ejecutar el bloque de código hasta que esta no se cumpla



## 4.3. Excepciones

Son estructuras que se ejecutan en caso de que ocurra un evento anormal que ocurre durante la ejecución del código, este tipo de estructuras solo está disponible para algunos lenguajes como C# y Java.

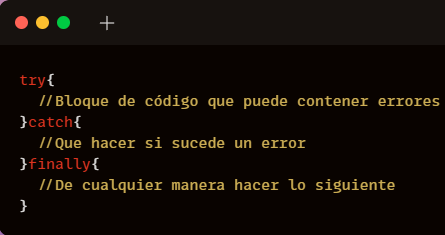
Los eventos anormales más comunes son desbordamiento de memoria, división por 0…

### 4.3.1 Throw

La podemos invocar en el momento que deseemos. Es como un plan B que se ejecuta si ocurre un evento anormal

### 4.3.2 Try-Catch-Finally

Intenta hacer X pero si falla X has Y



# 5. Funciones

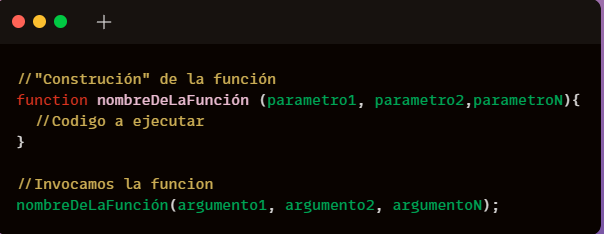
Son bloques de código que realizan una actividad especifica, nuestros programas más complejos eventualmente se construirán a base de varias funciones. Podemos pensar a las funciones como bloques que nos juntos nos permiten construir nuestro programa. El concepto de funciones está muy arraigado al de las matemáticas

¿Para qué nos sirven?

* Modularizar
* Optimizar
* Organizar
* Encapsular

Las funciones nos permiten hacer más escalable el código es decir a medida que crece nuestro programa sigue siendo igual de fácil depurarlo y entenderlo por nosotros y nuestro equipo de trabajo, como buena práctica es importante de que nuestra función se encargue de cosas puntuales

El principal atractivo de una función dentro de la programación es que puedes reutilizar el código es decir puedes volver a hacer lo mismo en cualquier parte del programa sin tener que volver a escribir el mismo código

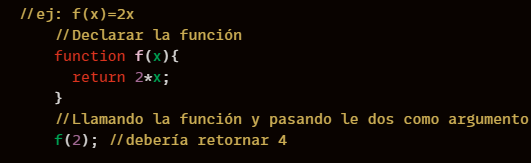


## 5.1 Parámetros vs Argumentos

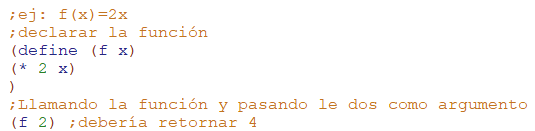
***Parámetros***: Se llaman así a el nombre de variable que le asignamos a esos datos de entrada cuando estamos declarando (“construyendo”) nuestra función

***Argumentos***: Se llaman así a los datos que le pasamos a nuestra función cuando la invocamos

Ejemplo en PHP con la función matemática

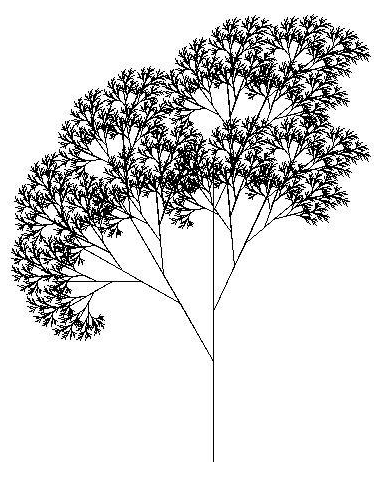


Ejemplo en scheme/racket con la función matemática



**Concepto de funciones auxiliares**: Las funciones auxiliares son funciones que son invocadas por otra principal estas son utilizadas cuando nuestros programas son muy grandes y complejos y debemos modularizarlos aún más.

# 6.Recursividad o recursión

La Recursión es: “Definir algo en términos de sí mismo”, “Una construcción que se construye de sí misma”

Usualmente cuando nos referimos a recursividad nos referimos a recursividad de funciones, pero también existen tipos de datos recursivos

## 6.1. Funciones recursivas

Son aquellas funciones que se llaman a sí mismas en el momento en que se están ejecutando.

El objetivo es que el problema se divida en varias (una o más) instancias del mismo problema, pero de tamaño menor gracias a lo cual se puede aplicar inducción, llegando a un punto donde se conoce el resultado (el caso base). Algunos autores llaman a esta estrategia “Divide y vencerás”

### Características de una función recursiva

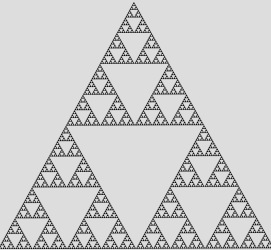
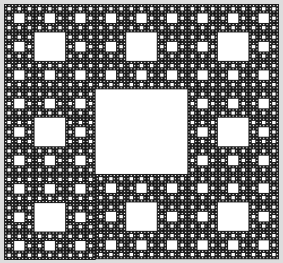
* Tienen uno o más **casos bases**, También llamados casos triviales, **criterio de parada o condición de parada**
* Tiene un caso recursivo, (La función se llama a sí misma)

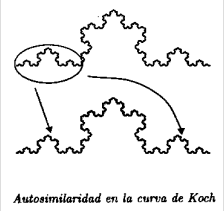
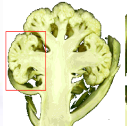
La estructura de una función recursiva consiste en una condicional **¿Cuántas condiciones debemos poner?** R/ una por cada caso base y una por cada caso recursivo

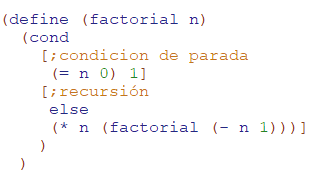
* Por cuestiones de calidad de software, un consejo es escribir primero las condiciones para los casos bases y de ultimo como un *else* el caso de recursión

### Ejemplos de recursión

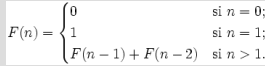
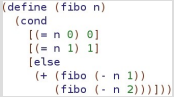
* Algoritmo de ordenamiento (de dos en dos)
* Método de Newton (para las raíces)
* El Máximo Común Divisor
* Torres de Hanoi
* Algoritmo Euclidiano
* Los fractales
  + El triángulo y la alfombra de Sierpinsky

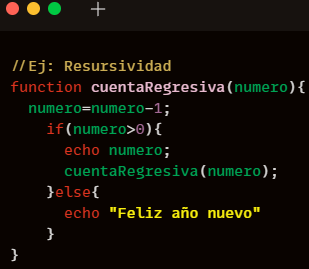
* + Curva de Koch
  +  
* Los números factoriales (Lenguaje: scheme/racket/lisp)
  + Definición matemática: imagen 1 y en scheme imagen 2:

1.  ****

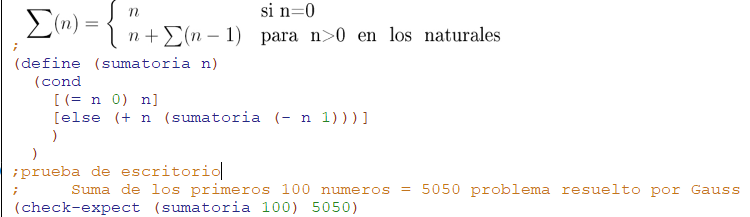
* La serie Fibonacci (Lenguaje: scheme/racket/lisp)
  + Definición matemática: imagen 1 y en scheme imagen 2:

1.  

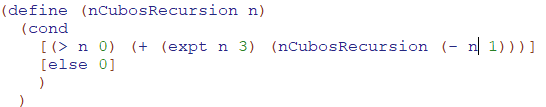
* cuenta regresiva en (lenguaje: PHP)



* Sumatoria (Lenguaje: scheme/racket/lisp)



* Sumatoria de los primeros n-cubos es decir (lenguaje: racket/scheme/lisp)



* Algoritmo general de la división

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Eficiencia de la recursividad

Por cada caso o llamada que se hace a la función el caso anterior queda esperando en la memoria (ocupa un espacio de memoria) esto hace que un ciclo sea más eficiente, por lo que siempre que podamos resolver un problema mediante un ciclo será la mejor opción

Sin embargo, muchos de los ejemplos mostrados anteriormente los hubiéramos podido hacer con un ciclo o loop, pero lo más importante es comprender el concepto ya que existen algunos problemas computacionales que solo se pueden resolver mediante recursividad

Fuentes: <https://es.wikipedia.org/wiki/Recursi%C3%B3n>

## 6.2Tipos de datos recursivos

Elementos como las listas enlazadas pueden ser un tipo de dato recursivo porque están compuestas a su vez de otras listas hasta llegar al caso base que es la lista vacía

### 6.2.1. Funciones recursivas en tipos de datos recursivos

A su vez con estos tipos de datos también podemos utilizar funciones recursivas para recorrerlos. En estos casos para aplicar la recursividad utilizaremos el mismo concepto, un condicional con el caso base, en este caso, nuestro caso base es la lista vacía es decir (empty? <lista>) = True, y nuestros casos recursivos dependerán del ejercicio invocando otra vez nuestra función.

A continuación, un ejemplo de cómo contar la cantidad de elementos de una lista. es decir, el equivalente a la función (lenght <list>) pero con recursividad

Interfaz de usuario gráfica, Texto, Aplicación

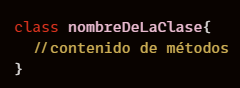
Descripción generada automáticamente

# 7.Introducción a la Programación Orientada a Objetos (POO)

Es un paradigma de programación utilizado por algunos lenguajes como: Java y C#

## 7.1 Clases

Es una plantilla para la creación de objetos según un modelo predefinido



# 7.2 Funciones vs métodos

Aunque suelen confundirse, si bien ambas estructuras se forman igual, pero están definidos en contextos diferentes, las funciones las podemos definir a fuera de las clases de una manera más “Libre” mientras que los métodos siempre van dentro de una clase

Alcance de métodos y sus diferencias

En los métodos también verás un prefijo interesante durante la definición de un método, estos son: **public, private, protected**

Cuando los encuentras sabrás que hace referencia a la visibilidad que este método tendrá dentro y fuera de la clase y además del alcance que pueda llegar a tener

Esto significa que, si defines un método **public**, podrá ser llamado y accedido desde otras clases que lo requieran, pero si es **private** solo lo podrás implementar dentro de la misma clase donde a sido definido y no será visible ni accesible desde afuera.

# 8. Estructuras de datos (Tipos de datos abstractos)

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. En esta descripción también entran los tipos de datos anteriormente nombrados, pero no nos vamos a centrar en los tipos de datos, sino en la forma de crear estructuras organizando estos tipos de datos compuestos.

## 8.1. Árbol

Es un tipo de dato basado en la teoría de grafos, se podría decir que es una estructura recursiva por que se compone de otros árboles. Se representa como un conjunto de nodos enlazados. Para que un árbol sea árbol un nodo hijo debe desprenderse de un único padre y no pueden existir ciclos

### 8.1.1. Terminología de los árboles

* Raíz: El nodo superior de un árbol
* Hijo: Un nodo conectado directamente con otro cuando se aleja de la raíz.
* Padre: La noción inversa de hijo
* Hoja: Un nodo sin hijos.
* Camino: Una secuencia de nodos y brazos conectados con un nodo
* Subárbol: El conjunto de los predecesores de un nodo

Forma

Descripción generada automáticamente

### 8.1.2. Usos comunes de los arboles

* Representar un dato jerárquicamente
* Almacenar un dato de tal modo que su búsqueda sea eficiente
* Representar listas ordenadas de datos
* Como un flujo de trabajo para la composición de imágenes digitales
* Algoritmos de encaminamiento

### Tipos de árboles

* Árbol binario: Es un árbol que crece a un factor de dos ramas, es decir que todos los nodos excepto las hojas tienen una rama izquierda y una rama derecha
* Árbol n-ario: Es un árbol que crece a un factor de n ramas donde ()
* Árbol balanceado: es aquel de su raíz se desprenden la misma cantidad de nodos tanto por izquierda como por derecha
* Árbol ordenado: Con ordenado nos referimos a que en el hijo izquierdo de cada nodo se encontrarán índices menores que el índice padre, mientras que en el hijo derecho de cada nodo se encontrarán valores de índice mayores o iguales que el valor de índice del padre

### 8.1.4. Representaciones (//):

En podemos representar árboles usando estructuras ó listas, Cuando los nodos tienen una cantidad restringida de hijos (no más de 2, 3, etc) se pueden usar estructuras Cuando la cantidad de hijos de un nodo no es restringida, es más apropiado usar. En los últimos nodos (hojas) los dos campos de izquierda y derecha los llenamos con .

Para hacer la representación por medio de listas y estructuras vamos a tomar como ejemplo el siguiente árbol:

Forma

Descripción generada automáticamente con confianza baja

#### 8.1.4.1 Representación con listas

Para representar cada nodo del árbol vamos a utilizar una lista cuyo primer elemento sea el valor, su segundo elemento será su hijo izquierdo y el tercer elemento será su hijo derecho



El árbol anterior lo representaríamos como:

Texto

Descripción generada automáticamente

#### 8.1.4.2 Representación con estructuras

Para representar cada nodo debemos definir una estructura, que tenga tres campos, 1. valor del nodo, 2. hijo izquierdo, 3. hijo derecho;



Utilizando estos nodos, nuestro árbol quedaría de la siguiente manera

Texto, Escala de tiempo

Descripción generada automáticamente

### 8.1.5 Operaciones básicas:

Las siguientes operaciones las necesitaremos como funciones auxiliares para la mayoría de los ejercicios que trabajemos con árboles y recursión, a continuación, las tres operaciones básicas para recorrer un árbol para trabajar con estructuras y con listas

#### 8.1.5.1 Función para Retornar el hijo izquierdo de un nodo:

Texto

Descripción generada automáticamente

#### 8.1.5.2 Función para retornar el hijo derecho de un nodo:

Texto

Descripción generada automáticamente

#### 8.1.5.3 Función para saber si un nodo es una hoja:

Esta función nos servirá como caso base cuando hagamos recursión, devuelve si el nodo actual es una hoja y de lo contrario

Imagen que contiene Texto

Descripción generada automáticamente

### 8.1.6. Ejercicios básicos con arboles

Comúnmente nos encontraremos con dos tipos de ejercicios con árboles

1. Buscar un nodo dentro del arboles por un índice y recuperarlo
2. Aplicar una operación a todos los nodos

Cada tipo de estos ejercicios se realiza de forma distinta, por lo que lo veremos por separado. Obviamente estos ejercicios se pueden complementar

#### 8.1.6.1 Ejercicios de búsqueda

Los ejercicios de búsqueda consisten en buscar un nodo en un árbol teniendo un índice que estamos utilizando para encontrarlo. Comúnmente en estos ejercicios de búsqueda utilizamos *ABB* (arboles de búsqueda binaria) que no es más que un árbol binario que se encuentra ordenado (Mediante algún índice que escojamos [puede ser un código]) y preferiblemente balanceado

Sabiendo que un árbol ordenado consiste en que por cada nodo en su hijo izquierdo se encuentra otro nodo con índice menor mientras que a su lado derecho se encuentra un hijo con los índice mayor o igual y este patrón se repetirá desde la raíz hasta las hojas del árbol, la forma de búsqueda que haremos será por medio de comparar el índice del nodo actual con el del índice buscado para saber en que hijo seguir buscando (izquierdo o derecho) o si ya hemos encontrado el nodo que necesitamos. Este método lo podemos ver de ejemplo:

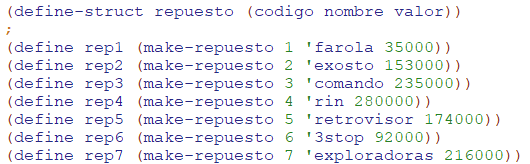
Imagen que contiene Escala de tiempo

Descripción generada automáticamente

Veamos el siguiente ejercicio donde resolvemos un ejercicio similar

##### 8.1.6.1.1 Ejemplo

Vamos a crear un árbol donde el valor de cada nodo consiste en una estructura como las siguientes:



Para facilitar nuestro recorrido vamos a hacer un árbol binario con listas, el árbol estará balanceado y ordenado por medio de su índice (código). Con ordenado nos referimos a que en el hijo izquierdo de cada nodo se encontrarán índices menores que el índice padre, mientras que en el hijo derecho de cada nodo se encontrarán valores de índice mayores o iguales que el valor de índice del padre. Siguiendo estos lineamientos nuestro árbol quedaría así:

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

En se representaría teniendo en cuenta las variables que ya habíamos definido de la siguiente manera:

Texto

Descripción generada automáticamente

Ahora la parte más importante de nuestro ejercicio es hacer una función que recorra el árbol para hacer alguna operación, por ello proponemos lo siguiente:

Hacer una función que pasándole el código y el árbol de repuestos me retorne el nombre del repuesto que corresponde a ese código, si no existe el código devolver

La calve para recorrer nuestra función es saber que es un árbol binario y ordenado por lo tanto establecemos las condiciones según esto, nuestra función quedaría así:

Texto, Carta

Descripción generada automáticamente

#### 8.1.6.2 Ejercicios de aplicar una operación a cada nodo

En este tipo de ejercicio generalmente utilizamos arboles binarios recorridos siguiendo un orden de recorrido, la elección de cada uno depende de los requerimientos de cada ejercicio. Los órdenes de recorrido son:

* ***Preorden***: 1. Raíz, 2. izquierdo y 3. derecho
* ***Inorden***: 1. Izquierdo, 2. Raíz, y 3. derecho
* ***Postorden***: 1. izquierdo, 2. Derecho, y 3. Raíz

En general la siguiente sería la forma, en esta imagen se encuentra:

Texto, Escala de tiempo

Descripción generada automáticamente

Veamos el siguiente ejemplo:

##### 8.1.6.2.1 Ejemplo

Tenemos que devolver una estructura donde sus elementos este ordenados de menor a mayor por medio de su índice, para ello vamos a utilizar un recorrido Inorden (Este ejemplo utiliza más funciones auxiliares solo que no las incluimos)

Texto

Descripción generada automáticamente

#### 8.1.6.3 Ideas de ejercicios para repasar

* Enumerar todos los elementos
* Enumerar la sección de un árbol
* Buscar un elemento
* Añadir un nuevo elemento en una determinada posición del árbol
* Borrar un elemento
* Podar: Borrar una sección entera de un árbol
* Injertar: Añadir una sección entera a un árbol
* Buscar la raíz de algún nodo
* Representar cada nodo como una variable en el montículo con punteros.
* Representar el árbol con un vector