RECURSION

La recursión es una técnica para programar computadoras que involucra el uso de procedimientos, subrutinas, funciones o algoritmos que se llaman a si mismos hasta alcanzar una condición de finalización.

La **recursión** es el proceso en el cual una función se invoca a sí misma. Este proceso permite crear un nuevo tipo de ciclos.

Función recursiva: Una función recursiva es cualquier función que se llama a sí misma.

Cualquier función recursiva tiene dos secciones de código claramente divididas:

● **Caso base:** tiene que existir siempre una condición en la que la función retorna sin volver a llamarse. Es muy importante porque de lo contrario, la función se llamaría de manera indefinida.

● **Caso recursivo:** sección en la que la función se llama a sí misma, con alguna variación en sus argumentos.

SCOPE DE VARIABLES

Se refiere al área del programa donde las variables del programa pueden ser accedidas después de haber sido declaradas.

De manera general, podemos decir que hay dos lugares donde las variables pueden ser declaradas:

● Las **variables locales** se declaran dentro de una función o un bloque de código.

● Las **variables globales** se declaran por fuera de una función

Algoritmos recursivos y algoritmos iterativos:

● Llamaremos algoritmos recursivos a aquellos que realizan llamadas recursivas para llegar al resultado.

● Llamaremos algoritmos iterativos a aquellos que llegan a un resultado a través de una iteración mediante un ciclo definido o indefinido.

Todo algoritmo recursivo puede expresarse como iterativo y viceversa

POO

Paradigma Imperativo: Un lenguaje imperativo permite escribir programas como un conjunto de instrucciones que se ejecutan una por una, de principio a fin, de modo secuencial excepto cuando intervienen instrucciones de salto de secuencia o control. El concepto central para la

ejecución del programa es el concepto de estado, que va modificándose a medida que asignamos y cambiamos valores a variables

Paradigma Orientado a objetos: Sintaxis alternativa, que nos permite presentar de forma más concisa la estructura de nuestro programa.

● Los programas se componen de definiciones de objetos y definiciones de funciones, y la mayor parte de la computación está definida en términos de cambios o modificaciones a dichos objetos.

● Cada definición de objeto se corresponde con algún objeto o concepto del mundo real, y las funciones que operan en dicho objeto se corresponden con alguna interacción entre objetos en el mundo real.

● El concepto central sigue siendo el de estado, pero la diferencia es que ahora además del estado global del programa, cada objeto tiene un estado propio

Programación Orientada a Objetos:

Nos permite organizar el código de manera que se asemeje a como pensamos en la vida real. Así, pensamos en los problemas como un conjunto de objetos que se relacionan entre sí.

Estos objetos nos permiten agrupar un conjunto de variables (**atributos**) y funciones (**métodos**) relacionadas en un mismo espacio

Clase: Una **clase** es una plantilla para la creación de nuevos objetos.

Es un diseño que especifica cómo se deben estructurar y comportar los objetos que se crearán a partir de ella. En términos más simples, una clase define las propiedades y métodos comunes que los objetos creados a partir de ella compartirán.

Objeto: Es una entidad concreta que se crea en memoria según el diseño definido por su clase. Los objetos tienen características específicas (atributos o variables de instancia) y pueden realizar acciones (métodos) según lo que la clase les haya definido. Cuando un objeto pertenezca a una clase, diremos que ese objeto es una **instancia** de la clase.

Método: Un método es una función que pertenece a una clase o a un objeto y define su comportamiento o acción específica. Es una función asociada a los objetos de una clase y puede acceder y manipular los datos que pertenecen a esos objetos. Los métodos representan el comportamiento de los objetos y se definen dentro de la clase mediante la palabra clave `def`

Composicion:

Que un operador sea composicional quiere decir que podemos acceder a sus atributos de manera animada. Por ejemplo, para obtener la coordenada x de la esquina de un rectángulo podemos obtenerla con: rectángulo.esquina.x

Mutabilidad:

Los objetos que pasamos a una función pueden ser alterados dentro de una función y dichos cambios sobreviven, aun cuando el marco de ejecución de dicha función haya terminado. Ejemplo, una función agrandar_rectángulo

● Las funciones puras no modifican ningún objeto pasado a las mismas.

● Las funciones modificadoras son aquellas que realizan cambios en los objetos pasados a las mismas

METODOS CONSTRUCTORES Y SOBRECARGA DE OPERADORES:

El método constructor __init__ será llamado internamente por Python cada vez que instanciemos un nuevo objeto de una clase. El constructor hace explícito qué atributos maneja la clase

Métodos especiales o métodos mágicos: Son métodos que, sí están definidos para un objeto, Python se encargará de invocarlos ante ciertas circunstancias. Podemos sobrecargar cualquier operador aritmético o de comparación

HERENCIA:

La herencia es la habilidad de definir una nueva clase que es una versión modificada de una clase ya existente.

● La principal ventaja de esta funcionalidad es que puedes añadir nuevos métodos a una clase sin modificar la clase ya existente

● Es llamada “herencia” porque la nueva clase hereda todos los métodos de la clase que ya existe. Esta metáfora se extiende y la clase que ya existía se suele llamar clase padre. La nueva clase se llama, entonces clase hija o subclase.

● La estructura de la herencia incluso refleja la naturaleza del problema, haciendo el programa más fácil de entender

TADs:

Un **tipo abstracto de datos** o TAD es:

● Una **colección de datos**

● acompañada de un **conjunto de operaciones o métodos y la semántica de dichas operaciones** (que hacen), pero no da detalles sobre las implementaciones de las mismas. Esto es lo que la convierte en abstracto

Son útiles porque..

- Los programas que los usan hacen referencia a las operaciones que tienen, no a la representación, y por lo tanto ese programa sigue funcionando si se cambia la representación.

- Simplifican el desarrollo de algoritmos utilizando las operaciones del tipo abstracto de dato, sin importar cómo las mismas son implementadas

- Algunos TADs utilizados con frecuencia, son implementados en librerías estándares de manera que puedan ser utilizadas por cualquier programador

- Las operaciones de los TADs proveen una especie de lenguaje de alto nivel para discutir y especificar otros algoritmos

Un **tipo de datos abstracto** o TAD es un tipo de datos que está definido por las operaciones que contiene y cómo se comportan (su interfaz), no por la forma en la que esas operaciones están implementadas.

Una **estructura de datos** es un formato para organizar un conjunto de datos en la memoria de la computadora, de forma tal de que la información pueda ser accedida y manipulada en forma eficiente

● Una estructura de datos contigua es aquella que, al representarse físicamente en la memoria del computador, lo hacen situando sus datos en áreas adyacentes de la memoria, es decir, los datos se ubican uno al lado del otro.

● Una estructura de datos enlazada es aquella donde los datos pueden estar desperdigados en la memoria, y hay una estructura adicional, llamada puntero que se usa para relacionar los datos entre sí.

Lista Enlazada

Una lista enlazada está formada por una secuencia de nodos, en la que cada nodo contiene un dato y una referencia al nodo que le sigue.

→ Es de **tiempo constante** insertar o eliminar elementos, pero es de **tiempo lineal** acceder a un elemento en particular, ya que es necesario pasar por todos los anteriores para llegar a él

Las operaciones que tiene son:

str

len

append

insert

remove

pop

index

Invariantes de objeto:

Los invariantes son condiciones que deben ser siempre ciertas. Las invariantes de objeto son invariantes que deben ser ciertas a lo largo de toda la existencia del objeto.

En la clase **ListaEnlazada** son los atributos **prim** y **len**

→ Cuando se desarrolla una estructura de datos como la lista enlazada, es importante destacar cuáles serán sus invariantes, ya que en cada método habrá que tener especial cuidado de que los invariantes permanezcan siempre ciertos.

TAD PILA

Su comportamiento se basa en “Lo último que se apiló es lo primero que se usa” ya que el último elemento en entrar será el primero en salir. Permite agregar elementos y sacarlos en el orden inverso al que se los colocó, de la misma forma que una pila (de platos, libros, etc)

Interfaz:

Push

Pop

isEmpty

TAD COLA

Modela el comportamiento: “el primero que llega es el primero en ser atendido”. Permite agregar elementos y sacarlos en el mismo orden en que se los colocó, como una cola de atención en la vida real. Los elementos se van encolando hasta que les toque su turno.

Interfaz:

Insert

Remove

isEmpty

TAD COLA PRIORIDAD

Misma interfaz que el TAD cola pero diferente semántica:

El elemento que se elimina de la cola no es necesariamente el primero que se agregó. Más bien, es el elemento en la cola que tiene la máxima prioridad el cual se elimina. Cuáles son las prioridades y cómo se comparan unas con otras no está especificado por la implementación de Cola de Prioridad. Depende de cuáles sean los elementos que estén en la cola.

ARBOLES

Un **árbol T** es un conjunto de puntos a los que llamamos vértices, que pueden estar unidos con líneas, las cuales llamamos aristas, que satisface que: si **v** y **ω** son vértices en **T**, existe un **único camino** desde **v** a **ω.**

Un árbol con raíz es un árbol en el que un vértice específico se designa como raíz.

ARBOLES BINARIOS

Un **árbol binario** es un árbol con raíz en el que cada vértice **tiene ningún hijo, un hijo o dos hijos**. Si el vértice tiene un hijo se designa como un hijo izquierdo o como un hijo derecho (pero no ambos). Si un vértice tiene dos hijos, un hijo se designa como hijo izquierdo y el otro como hijo derecho

Un **árbol binario completo** es un árbol que **tiene cero o dos hijos**

Los árboles son **estructuras de datos recursivas** porque se definen recursivamente.

Un árbol binario es:

● El árbol vacío, representado por None, o

● un nodo que contiene una referencia de objeto y dos referencias a árboles binarios.

ARBOLES BINARIOS DE BUSQUEDA

Un **árbol binario de búsqueda** es un árbol binario T en el que se asocian datos a los vértices. Los datos están arreglados de manera que, para cada vértice v en T, cada dato en el subárbol de la izquierda de v es menor que el dato en v, y cada dato en el subárbol de la derecha de v es mayor que el dato en v.

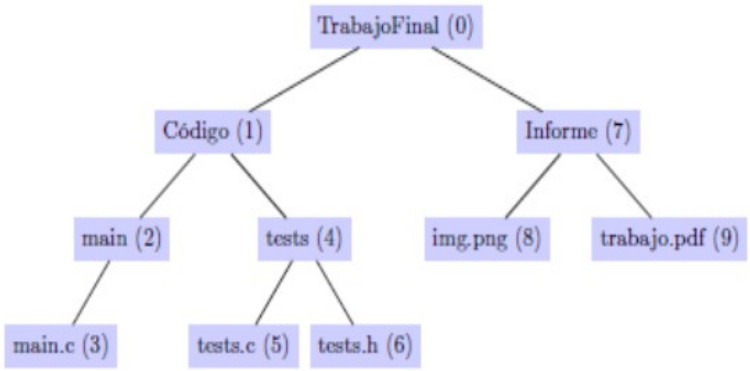
Recorridos de árboles:

PreOrder:

1. Visite la raíz

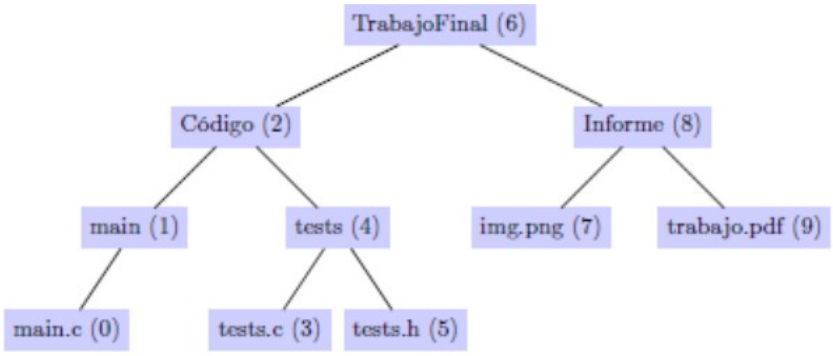
2. Recorra en PreOrder el sub-árbol izquierdo

3. Recorra en PreOrder el sub-árbol derecho



InOrder:

1. Recorra InOder el sub-árbol izquierdo
2. Visite la raíz
3. Recorra en InOrder el sub-árbol derecho



PostOrder:

1. Recorra en PostOder el sub-árbol izquierdo
2. Recorra en PostOder el sub-árbol derecho
3. Visite la raíz